The RSA Public Key Cryptosystem

William P. Wardlaw

Mathematics Department, U. S. Naval Academy, Annapolis, MD, 21146

1 Introduction

The RSA (Rivest, Shamir, Adleman) cipher algorithm has captured the imagination of many mathematicians by its elegance and basic simplicity ever since it was introduced in 1978. Numerous descriptions of the algorithm have been published. Readers with a knowledge of a little basic number theory will find the original paper [RSA] by the inventors of the algorithm, Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, quite readable. Perhaps the most famous description is Martin Gardner's expository article [G], which is written for readers of *Scientific American.* Martin E. Hellman [H] wrote another good *Scientific American* article describing the RSA algorithm and the knapsack cipher algorithm. The goal of this paper is to lead the reader who has some mathematical maturity but no knowledge of number theory, say a first year calculus student, a clever high school student, or an interested engineer, through the basic results needed to understand the RSA algorithm. The prerequisites are only a knowledge of the elementary school arithmetic of the integers, high school algebra, some familiarity with the notions of sets and of functions, and, most importantly, a real desire to understand how the RSA algorithm works. We begin with a discussion of general crypto systems and the differences between classical systems and public key systems. Then the treatment will give an informal but fairly rigorous introduction to the division algorithm, divisibility properties, greatest common divisors, the Euclidean algorithm, modular arithmetic, repeated squaring algorithm for $b^a \pmod m$, time estimates for these algorithms, Euler's totient function, Euler's Theorem, and, as a corollary, Fermat's Little Theorem. Don't worry if you are unfamiliar with some of these terms now - they will all be explained when they arise. These ideas will then be used to explain the RSA algorithm in detail. We will mention but not go into detail on the notions of primality testing and methods of factoring. The student who wishes a deeper understanding of these things is strongly recommended to read the pertinent sections of Neal Koblitz' excellent book [Ko], *A Course in Number Theory and Cryptography.*

2 General Cryptosystems

A **cryptosystem** is a method of secret communication over public channels between members of some group of people, which we call the **crypto group**.

The term **public channels** refers to the possibility that people outside of the crypto group can intercept messages sent between members of the group. Broadcast radio, telephone lines, ordinary mail, and e-mail are all examples of public channels. A cryptosystem will be made up of one or more (usually many) units, called **crypto cells**, each of which provides for communication from one member of the group to another.

Suppose that Bob wants to send a message x to Alice. Bob uses an **encryptor** E to act on x and transform it to the encrypted message $y = xE$. Then he sends the encrypted message y to Alice. When Alice receives the message, she uses a **decryptor** D to convert y back to the plaintext message x: $yD = (xE)D = x$. Thus, the decryptor D undoes or **inverts** the action of the encryptor E. In practice, E (as well as D) might be a mechanical device, a computer program, or an algorithm) which converts x into y (or y into x, in the case of D). This encryptor-decryptor pair (E, D) is the simple cryptosystem, or crypto cell, that Bob uses to contact Alice.

We model this situation mathematically by taking E to be a one to one function with domain $X = dom(E)$ and range $Y = ran(E)$, and taking $D = E^{-1}$ with domain $Y = dom(D)$ and range $X = ran(D)$ to be the inverse function of E. Thus, X is the set of all plaintext messages that can be encrypted by E, and Y is the set of all encrypted messages. The functions E and D satisfy the relationship

$$xE = y \text{if and only if } yD = x. \tag{1}$$

A simple example is given by letting $X = Y = \{a, b, c, d, ..., x, y, z\}$ be the alphabet and taking E to be the permutation

$$E = \begin{pmatrix} \text{a b c d e f g h i j k l m n o p q r s t u v w x y z} \\ \text{k r y f m t a h o v c j q x e l s z g n u b i p w d} \end{pmatrix} \tag{2}$$

Now E acts on a letter in the top row by transforming it to the letter below it in the second row. Thus, aE = k, bE = r, ... , and zE = d. It is not difficult to construct $D = E^{-1}$ from E to obtain

$$D = \begin{pmatrix} \text{a b c d e f g h i j k l m n o p q r s t u v w x y z} \\ \text{g v k z o d s h w l a p e t i x m b q f u j y n c r} \end{pmatrix} \tag{3}$$

It may seem silly to let every message consist of a single letter, but we do not need to stretch our imagination far to see how E encrypts "to err is human" as "ne mzz og huqkx" and D decrypts "ne tezaobm foboxm" as "to forgive divine". The encryptor E is an example of an important class of $26! = 1 \times 2 \times 3 \times ... \times 25 \times 26$ encryptors called **monoalphabetic ciphers** that are really just permutations of the alphabet. It is a simplifying convenience to think of the set X of all plaintext messages as consisting of 26 letters,

rather than the infinite collection of words and phrases that can be written with these letters!

As we mentioned above, a cryptosystem consists of crypto cells (E, D) which allow one member of the group to send a private message to another. In a *classical* cryptosystem, the situation is as described in our example. If Bob knows an encryptor E to send a message to Alice, he can easily figure out the corresponding decryptor D, and so he can use D to understand messages that Alice sends to him using E. The situation is *symmetrical*; Alice and Bob share E and D, and use them to communicate. In order to set up private communication over a public channel, Bob and Alice would first have to get together privately and share the information (E, D), usually in the form of a "key". Unfortunately, the problem of key exchange is made worse by the fact that if Bob and Alice use the same key for too long, an easedropper may be able to break their key and decrypt their communications, so Bob and Alice must get together frequently to exchange keys. David Kahn describes the following consequences of not changing keys in Chapter 17 of [K] , especially page 567. After their Pearl Harbor attack, the tremendous success of the Japanese in spreading their forces throughout the Pacific delayed their intended change of codebooks from 1 April to 1 June 1942. This enabled the cryptanalysts in Hawaii to glean enough information from Japanese coded messages to predict the Japanese attack on Midway and to get U. S. carriers in the right place to surprise the Japanese and win a decisive victory.

Because of this need to avoid sending too many messages using the same crypto cell (E, D), a cryptosystem, even one involving only two people, will often use a large number of different crypto cells. Each crypto cell will be given an identifying key. Different messages will then be encrypted using different encryptors, with keys determined by the date, the time of day, or somehow hidden in the message itself.

Another reason that a crypto system may require a number of different crypto cells is to establish different *cliques* in the crypto group. A **clique** within a crypto group is a set of people within the group who exclusively share a given set of crypto cells. Any pair of a clique can communicate with each other without members of other cliques being able to decrypt their messages. In a military crypto group, a clique of all officers might have access to those crypto cells used to send "confidential" messages, but only commissioned officers would be allowed to read "secret" messages, and a still smaller clique of officers with a special clearance would have access to the crypto cells used for "top secret" traffic.

One useful organization of a crypto group is to make each pair a clique, so that any two members of the group can communicate secretly with each other. In general, a crypto group of N people would have $N(N-1)/2$ pairs, and it would require that many crypto cells (E, D) in order to allow any two people in the group to enjoy private communication. For example, in a group of 10 people, we have 10 ways to pick the first person x in an ordered pair (x, y), and for each such x there would be 9 choices for y, for a total of

10×9 *ordered* pairs of people. Since this method counts each *unordered pair* {x, y} twice, there are $10 \times 9/2 = 45$ unordered pairs.

For several thousand years, only classical (also called symmetrical) crypto systems were available. But in 1976, Diffie and Hellman [DH] introduced the idea of a **public key cryptosystem**. In such a system, each user secretly obtains a crypto cell (E, D) and then *publishes* the encryptor E. Clearly, the central requirement of such a system is that it be *prohibitively difficult* to figure out the decryptor $D = E^{-1}$ from a knowledge of E. For example, if a given computer takes a millisecond to encrypt a given message using the encryptor E, that same computer might take several thousand years to compute the decryptor D using a knowledge of E. (This is called **breaking** the encryptor.) Of course, several thousand computers working together might break the encryptor in only one year, and a single computer which computes several million times as fast as the original computer might break the encryptor in nine or so hours. It is clear that the security of the system is dependent on the present state of technology.

Diffie and Hellman suggested the use of a **trapdoor function** E for which the possession of certain secret information would make it easy to calculate its inverse D, but for which D would be very difficult to discover without this information. Then each member M of a crypto group would calculate and publish an encryptor E_M and privately calculate its inverse, the decryptor D_M, which he would keep secret. If Bob wants to send a message x to Alice, he needs no private meeting to exchange keys. He simply looks up Alice's published encryptor E_A, uses it to encrypt the message, and sends the encrypted message $y = xE_A$ to Alice. Since only Alice knows her decryptor D_A, only Alice can decrypt y to obtain $x = yD_A = xE_AD_A$. If she wishes to reply, Alice can use Bob's published encryptor E_B.

But how can Alice be sure that it was Bob who sent her the message? After all, everyone has access to her encryptor E_A and could use it to send her a message masquerading as Bob. But authentification is possible at the cost of two extra messages. Bob could append a ten digit random number b as part of his first message. Alice could generate a ten digit random number a and send Bob the number $a + b$, using the encryptor E_B that only Bob can decrypt. Bob then authenticates his original message as well as future messages by appending a, which he obtains by subtracting b from Alice's appended $a + b$.

For certain public key crypto systems, Bob and Alice can even sign their messages in a way that can be verified by an arbiter later. Let $X_M = dom(E_M) = ran(D_M)$ be the set of all plaintext messages and $Y_M = ran(E_M) = dom(D_M)$ be the set of all encrypted messages for the crypto cell (E_M, D_M) belonging to crypto group member M. Suppose that every plaintext message can be considered to be an encrypted message, and vice versa. That is, $X_M = Y_M$ for every member M. If $X_A = Y_A \subseteq X_B = Y_B$, then Bob can send a signed message x in X_A to Alice by first encrypting it using Alice's encryptor E_A to obtain $y = xE_A$ in $X_A = Y_A$. Since $Y_A \subseteq Y_B$, y is also in Y_B,

and so Bob's decryptor can be applied to y to obtain $z = yD_B = xE_AD_B$ in X_B. Then Bob sends the signed encrypted message z to Alice. When Alice receives the message z, she knows it is supposed to have come from Bob and that $X_A \subseteq X_B$, so she first applies Bob's publicly known encryptor E_B and then her secret decryptor D_A to read the message $zE_BD_A = (xE_AD_B)E_BD_A = xE_AD_A = x$: E_B "undoes" D_B and then D_A "undoes" E_A. Alice knows the message was originated by Bob, because only Bob is able to use his secret decryptor D_B to construct his encrypted message z. If another decryptor had been inserted instead, then Alice's use of Bob's encryptor E_B would have produced gibberish instead of the intelligible plaintext message x. Only Bob could have sent the message! Moreover, if Bob later denies sending the message (which could be a contract of some sort), Alice can show the messages x, y, and z to the judge. The judge can then observe that $xE_A = y = zE_B$ to verify Alice's claim that Bob sent the message: Although Alice could make up x and construct $y = xE_A$, she could not construct $z = yD_B$ without knowing Bob's decryptor D_B. Bob must have sent the message!

Similarly, Alice can send a signed message u to Bob by first applying her decryptor D_A to produce $v = uD_A$ in $Y_A \subseteq X_B$, and then applying Bob's encryptor E_B to produce $w = vE_B = uD_AE_B$ in Y_B, which she sends to Bob. Bob then is able to decrypt and read $u = wD_BE_A$.

Note that the order in which the encryptors and decryptors are applied is important if the containment $X_A \subset X_B$ is strict and $X_A \neq X_B$. If instead of $z = xE_AD_B$ Bob tried to form $z' = xD_BE_A$, he would start with x in $X_A \subset X_B$ and apply D_B to obtain $y' = xD_B$ in X_B, but maybe *not in* X_A ! If y' is not in X_A, Bob cannot apply E_A to y' to get $z' = xD_BE_A$ because E_A only works on elements of its domain $dom(E_A) = X_A$. A similar problem could occur if Alice changed the order in which the encryptors and decryptors are applied in sending a signed message to Bob.

Here are some advantages of public key crypto systems over classical crypto systems:

(1) There is no need for private meetings to exchange keys.

(2) Only N crypto cells are needed for private communication between each pair of a crypto group of N people using a public key system, but $N(N-1)/2$ are needed for a classical system, an increase by a factor of $(N-1)/2$.

(3) Some public key systems allow signatures on messages.

The major disadvantage of public key cryptosystems is that those that have been invented so far are up to a thousand times slower in encrypting and decrypting messages. For this reason, a major use of public key systems may be to exchange keys for a faster classical system.

The first and still most popular public key cryptosystem is the RSA algorithm, which was introduced by its inventors, Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman in their 1978 paper [RSA].

3 Arithmetic

Now we look into what the RSA is and why it works, beginning with a close look into the arithmetic of integers. We will let $\mathbf{N} = \{1, 2, 3, ...\}$ be the set of positive integers or natural numbers and $\mathbb{Z} = \{0, -1, 1, -2, 2, ...\}$ be the set of all integers. (The $\mathbb{Z}$ comes from the German word "Zahl" for "number".) All of the numbers we discuss in this section will be integers; that is, elements of $\mathbb{Z}$. We assume the truth of the following axiom.

Theorem 1. *(***Well Ordering Axiom***) Every nonempty subset of nonnegative integers has a smallest element.*

In grade school you learned a method of long division of a positive integer a (the dividend) by another positive integer d (the divisor) which produced successive remainders and continued until the final remainder r was smaller than the divisor. This process actually proves the next theorem for positive integers a and d, and you can figure out from the positive case why it is true regardless of the signs of a and d. We give another demonstration using (1).

Theorem 2. *(***Division Algorithm***) If a and d are integers and $b \neq 0$, then there are unique integers q and r which satisfy*
(a) $a = dq + r$ *and*
(b) $0 \leq r < |d|$.

Proof. Consider the set S of all integers of the form $a - dx$, where x can be any integer. Since x can be large and positive or large and negative and $d \neq 0$, it is clear that S contains a nonnegative integer. Let $r = a - dq$ be the smallest nonnegative integer in S, obtained when $x = q$. Then $a = dq + r$, so (2a) holds. If $r \geq |d|$ we let $s = sgn(d) = \pm 1$ so that $ds = |d|$; then $x = q + s$ gives $0 \leq a - dx = a - d(q + s) = a - dq - ds = r - |d| < r$, contradicting the choice of r as the smallest nonnegative element of S. Thus, $0 \leq r < |d|$ and r satisfies (2b). To show uniqueness, we assume that $a = dq + r = dq' + r'$ and $0 \leq r' \leq r < |d|$. Then $0 \leq r - r' = d(q' - q) = |d||q' - q|$. It is clear that $r - r' \leq r < |d|$, so we must have $|q' - q| < 1$. Hence we must have $q' - q = r - r' = 0$, so $r = r'$ is unique and $q = q'$ is unique. □

When the remainder $r = 0$ and $a = dq$, we say that d **divides** a, or equivalently, a is a **multiple** of d, and we write $d \mid a$. Otherwise, d does not divide a and we write $d \nmid a$. To reiterate, $d \mid a$ if and only if there is an integer m such that $a = dm$. For example, $3 \mid 12$ and $6 \mid 18$, but $5 \nmid 12$ and $15 \nmid 18$. The following divisibility properties are easily shown to be true.

Lemma 3. *(***Divisibility Properties***) If a, b, c, d, x, and y are integers, then*

(a) $a \mid b$ *and* $b \mid c$ *implies that* $a \mid c$ *. (Divisibility is* **transitive***.)*
(b) $a \mid b$ *and* $b \mid a$ *implies that* $a = \pm b$ *. (Divisibility is* **antisymmetric***.)*

(c) $d \mid a$ *and* $d \mid b$ *implies that* $d \mid xa + yb$.

An integer d which divides both of the integers a and b is called a **common divisor** of a and b, and the largest of these (when a and b are not both 0) is called the **greatest common divisor (gcd)** of a and b. We write $d = \gcd(a, b)$ for this necessarily positive integer, and define $\gcd(0, 0) = 0$.

Theorem 4. *(***GCD Theorem***) If* a *and* b *are any integers, then there are integers* x *and* y *such that* $\gcd(a, b) = xa + yb$.

Proof. This is clear for $a = b = 0$. If either a or b is nonzero, let S be the set of all positive integers of the form $sa + tb$, where s and t are integers. Since S is not empty, it has a smallest element by (1). Let $d = xa + yb$ be the smallest element of S. The division algorithm (2) tells us that there are integers q and r satisfying $a = dq + r$ and $0 \leq r < d$. Now $r = a - dq = (1 - x)a + (-y)b$ would be in S if it were positive, which would contradict our choice of d as the smallest element of S. Therefore, $r = 0$ and $a = dq$ is divisible by d. Similarly, $d \mid b$, and so it is a common divisor of a and b. If c is another common divisor of a and b, then $c \mid d = xa + yb$ by (3c) and $d > 0$ implies $c < d$ and $d = \gcd(a, b)$. □

Now it is easy to see that $\gcd(a, 0) = |a|$ and $\gcd(a, b) = \gcd(|a|, |b|)$, so we only need to find gcds of positive integers. Note that if $0 < b < a$, we can use the division algorithm (2) to obtain $a = bq + r$ and $0 \leq r < b$. It is clear from (3c) that the common divisors of a and b are exactly the same as the common divisors of b and r, and so $\gcd(a, b) = \gcd(b, r)$. This suggests that a sequence of divisions can determine the gcd of two positive integers:

$$\begin{aligned}
&(0) && a = bq_1 + r_1 \\
&(1) && b = r_1 q_2 + r_2 \\
&(2) && r_1 = r_2 q_3 + r_3 \\
& && \vdots \\
&(k) && r_{k-1} = r_k q_{k+1} + r_{k+1} \\
& && \vdots \\
&(n) && r_{n-1} = r_n q_{n+1} + r_{n+1}
\end{aligned} \tag{4}$$

If we continue until the remainder $r_{n+1} = 0$ we will have $\gcd(a, b) = \gcd(b, r_1) = \gcd(r_1, r_2) = \gcd(r_k, r_{k+1}) = \gcd(r_n, 0) = r_n$. This method of finding the gcd was published by Euclid in his *Elements* more than 2000 years ago. It is called **the Euclidean algorithm**. Next we give an example with $a = 54321$ and $b = 12345$.

$$\begin{aligned}
(1)\ 54321 &= 12345 \times 4 + 4941 \\
(2)\ 12345 &= 4941 \times 2 + 2463 \\
(3)\ \ 4941 &= 2463 \times 2 + 15 \\
(4)\ \ 2463 &= 15 \times 164 + 3 \\
(5)\ \ \ 15 &= 3 \times 5 + 0
\end{aligned} \tag{5}$$

Thus $\gcd(54321, 12345) = 3$ is the remainder in row (4), the last remainder before the remainder becomes 0. In a later section we will see an extension of this algorithm which finds x and y such that $\gcd(a, b) = xa + yb$.

The divisors of 1 are called **units**. Actually, 1 and -1 are the only units among the integers. An integer a is a **composite** if there are integers b and c such that $a = bc$ and $1 < |b| \leq |c|$, that is, neither b nor c are units. The first ten positive composites are 4, 6, 8, 9, 10, 12, 14, 15, 16, and 18. An integer p is a **prime** if $p > 1$ and the only divisors of p are ± 1 and $\pm p$. The first ten primes are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Euclid presented a proof that every positive integer was a product of primes which was unique except for the order of the factors, and he showed that there were infinitely many primes. Two integers are said to be **relatively prime** if their only common factors are ± 1; that is, a and b are relatively prime if and only if $\gcd(a, b) = 1$.

4 Modular Arithmetic

Let m be an integer greater than 1. We say that integers a and b are **congruent modulo** m if and only if $m \mid a - b$, and we denote this by $a \equiv b \pmod{m}$. For example, $75 \equiv 9 \pmod{33}$ because $75 - 9 = 66 = 2 \times 33$ is divisible by 33. The relation of congruence mod m behaves pretty much like equality. The relation is an equivalence relation on the set $\mathbb{Z}$ of all integers which preserves addition and multiplication of integers.

(4.1) **Properties of congruence**: Let m be an integer greater than 1. Then, for any integers a, b, c, a', and b', the following properties hold:

(a) Reflexive: $a \equiv a \pmod{m}$.
(b) Symmetric: $a \equiv b \pmod{m}$ implies $b \equiv a \pmod{m}$.
(c) Transitive: $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m}$ implies $a \equiv c \pmod{m}$.
(d) Addition: $a \equiv a' \pmod{m}$ and $b \equiv b' \pmod{m}$ implies $a + b \equiv a' + b' \pmod{m}$.
(e) Multiplication: $a \equiv a' \pmod{m}$ and $b \equiv b' \pmod{m}$ implies $ab \equiv a'b' \pmod{m}$.

Proof. (a)-(d) are left to the reader. For (e), $a \equiv a' \pmod{m}$ and $b \equiv b' \pmod{m}$ implies $m \mid a - a'$ and $m \mid b - b'$ implies $m \mid (a - a')b + a'(b - b') = aa' - bb'$ implies $ab \equiv a'b' \pmod{m}$. □

For example, $37 \equiv 4 \pmod{33}$ and $45 \equiv 12 \pmod{33}$, so $37 + 45 = 82 \equiv 4 + 12 = 16 \pmod{33}$ and $37 \times 45 = 1665 \equiv 4 \times 12 = 48 \equiv 15 \pmod{33}$,

which can be seen directly by the facts that $33 \mid 82 - 16 = 66 = 2 \times 33$ and $33 \mid 1665 - 15 = 1650 = 50 \times 33$.

The Division Algorithm (2) shows that every integer a can be written in the form $a = mq + \overline{a}$ with $0 \leq \overline{a} < m$. Thus, every a can be reduced $(\mathrm{mod}\, m)$ to an integer $\overline{a}$ in the set $\mathbb{Z}_m = \{0, 1, 2, ..., m-1\}$ of **residues** modulo m. The example above illustrates that it is usually easier to reduce first and then perform arithmetic operations rather than the other way around!

Now we can consider only the m elements in $\mathbb{Z}_m$, perform addition and multiplication on these elements, and reduce them to again get elements of $\mathbb{Z}_m$. The result is an arithmetic modulo m on the set $\mathbb{Z}_m$ that is much the same as the arithmetic on $\mathbb{Z}$. But there are differences! For example, in $\mathbb{Z}_{33}$, $6 \times 9 \equiv 21 (\mathrm{mod}\, 33)$ and $6 \times 31 \equiv 21 (\mathrm{mod}\, 33)$, so the law of cancellation does not hold for multiplication by 6 modulo 33. Why not? We get a clue when we subtract the first of these congruences from the second: $6 \times (31 - 9) = 6 \times 22 = 132 \equiv 0 (\mathrm{mod}\, 33)$. Although $6 \not\equiv 0 (\mathrm{mod}\, 33)$ and $22 \not\equiv 0 (\mathrm{mod}\, 33)$, we still have $6 \times 22 \equiv 0 (\mathrm{mod}\, 33)$. But now we see why. $33 = 3 \times 11$; 6 has a factor of 3 and 22 has a factor of 11, so when multiplied together, the product has a factor of 33, and $33 \equiv 0 (\mathrm{mod}\, 33)$. Suppose we multiply by a number with no factor in common with 33, for example, 10. Then $10z \equiv 0 (\mathrm{mod}\, 33)$ means that $10z$ is divisible by 33. But then z must have both a factor of 3 and a factor of 11, since 10 has neither. That means that z is divisible by 33, and so $z \equiv 0 (\mathrm{mod}\, 33)$. Moreover, it follows that $10x \equiv 10y (\mathrm{mod}\, 33)$ implies $10(x - y) \equiv 0 (\mathrm{mod}\, 33)$ implies $x - y \equiv 0 (\mathrm{mod}\, 33)$ implies $x \equiv y (\mathrm{mod}\, 33)$. Multiplication by 10 is cancellative modulo 33. These facts are illustrated in the table below.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$6x$	0	6	12	18	24	30	3	9	15	21	27	0	6	12	18	24	30
$10x$	0	10	20	30	7	17	27	4	14	24	1	11	21	31	8	18	28

x	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
$6x$	3	9	15	21	27	0	6	12	18	24	30	3	9	15	21	27
$10x$	5	15	25	2	12	22	32	9	19	29	6	16	26	3	13	23

Multiplication of elements of $\mathbb{Z}_{33}$ by 6 gives all multiples of 3, each repeated 3 times. But multiplication by 10 gives every element of $\mathbb{Z}_{33}$ exactly once in another order. Note that 10 is a unit in $\mathbb{Z}_{33}$ because $10 \times 10 \equiv 1 (\mathrm{mod}\, 33)$.

Now let m be a fixed integer that is greater than 1, and let a be any integer. If $d = \gcd(m, a) > 1$, then $m' = m/d$ and $a' = a/d$ are both integers and $m' \not\equiv 0 (\mathrm{mod}\, m)$ since $0 < m' < m$ but $am' = a'm \equiv 0 (\mathrm{mod}\, m)$. If there is an $x \not\equiv 0 (\mathrm{mod}\, m)$ such that $ax \equiv 0 (\mathrm{mod}\, m)$ we call a a **zero divisor** modulo m.

On the other hand, if $\gcd(m, a) = xm + ya = 1$ by (3.4) then $ya \equiv 1 (\mathrm{mod}\, m)$ shows that a is a unit modulo m and a has an **inverse** $a^{-1} = y$. (This is why a unit is called **invertible**.) In this case, a is **cancellable** modulo

m, because the congruence $ax \equiv ay (\mathrm{mod}\, m)$ need only be multiplied on both sides by a^{-1} to show that $x \equiv y (\mathrm{mod}\, m)$. In particular, $az \equiv 0 (\mathrm{mod}\, m)$ implies $az \equiv a \cdot 0 (\mathrm{mod}\, m)$ implies $z \equiv 0 (\mathrm{mod}\, m)$, and so a is not a zero divisor. Thus we have the following result.

Theorem 5. *Let m be an integer that is greater than 1 and let a be any integer. Then the following are equivalent:*

(a) $\gcd(m, a) = 1$.

(b) $az \equiv 0 (\mathrm{mod}\, m)$ implies that $z \equiv 0 (\mathrm{mod}\, m)$; that is, a is not a zero divisor modulo m.

(c) $ax \equiv ay (\mathrm{mod}\, m)$ implies $x \equiv y (\mathrm{mod}\, m)$; that is, a is cancellable modulo m.

(d) a has an inverse modulo m; that is, there is an element b in $\mathbb{Z}_m$ such that $ab \equiv 1 (\mathrm{mod}\, m)$.

We define $\mathbb{Z}_m^* = \{a \in \mathbb{Z}_m : \gcd(a, m) = 1\}$ to be the set of units in $\mathbb{Z}_m$. If a is in $\mathbb{Z}_m^*$, then multiplication by a is cancellable modulo m, and so multiplying all of the elements in $\mathbb{Z}_m$ by a simply moves the elements around. Moreover, if a and b are both in $\mathbb{Z}_m^*$, then $abb^{-1}a^{-1} = a1a^{-1} = 1$, so ab is invertible and is also in $\mathbb{Z}_m^*$. That is, $\mathbb{Z}_m^*$ is closed under multiplication, and if a is any unit, multiplying the set $\mathbb{Z}_m^*$ of all units by a simply moves the elements of $\mathbb{Z}_m^*$ around, or permutes them. The following tables of multiples of $\mathbb{Z}_{33}^*$ illustrate this.

x	1	2	4	5	7	8	10	13	14	16	17	19	20	23	25	26	28	29	31	32
$7x$	7	14	28	2	16	23	4	25	32	13	20	1	8	29	10	17	31	5	19	26
$10x$	10	20	7	17	4	14	1	31	8	28	5	25	2	32	19	29	16	26	13	23

The **Euler totient function** φ is defined by $\varphi(m) = |\mathbb{Z}_m^*|$, that is, $\varphi(m)$ is the number of units modulo m. For example,

$$\varphi(33) = |\{1, 2, 4, 5, 7, 8, 10.13, 14, 16, 17, 19, 20, 23, 25, 26, 28, 29, 31, 32\}| = 20.$$

For general $m > 1$ and any a with $\gcd(m, a) = 1$, we can write $\mathbb{Z}_m^* = \{a_1, a_2, \ldots, a_{\varphi(m)}\} = a\mathbb{Z}_m^* = a\{a_1, a_2, \ldots, a_{\varphi(m)}\} = \{aa_1, aa_2, \ldots, aa_{\varphi(m)}\}$, since multiplication by a simply permutes the elements of $\mathbb{Z}_m^*$. Now let A be the product of every element in $\mathbb{Z}_m^*$. Then $A \equiv a_1 a_2 \cdots a_{\varphi(m)} \equiv aa_1 aa_2 \cdots aa_{\varphi(m)} \equiv a^{\varphi(m)} a_1 a_2 \cdots a_{\varphi(m)} \equiv a^{\varphi(m)} A \ (\mathrm{mod}\, m)$. Cancellation of A modulo m, which is valid since $A \in \mathbb{Z}_m^*$, gives $1 \equiv a^{\varphi(m)} (\mathrm{mod}\, m)$. This proves Euler's Theorem, which is the mathematical basis of the RSA algorithm!

Theorem 6. *(**Euler's Theorem**) If $\gcd(a, m) = 1$, then $a^{\varphi(m)} \equiv 1 \ (\mathrm{mod}\, m)$.*

Here are a couple of examples modulo 33: $\gcd(33, 10) = 1$ and $10^{20} = 100000000000000000000 = 99999999999999999 + 1 \equiv 1 (\mathrm{mod}\, 33)$;

$\gcd(33,7) = 1$ and $7^{20} = (49)^{10} \equiv (16)^{10} = (2^4)^{10} = 2^{40} = (2^5)^8 = (32)^8 \equiv (-1)^8 = 1 (\mathrm{mod}\, 33)$.

Now if p is a prime it is easy to see that $\varphi(p) = p - 1$; moreover, for any integer a, it is easy to see that $\gcd(a,p) = 1$ if $p \nmid a$ and $\gcd(a,p) = p$ if $p \mid a$. In the first case Euler's Theorem (4.3) shows that $a^{p-1} \equiv 1 (\mathrm{mod}\, p)$, and multiplication on both sides by a gives $a^p \equiv a (\mathrm{mod}\, p)$. In the second case, $a \equiv 0 \equiv a^p (\mathrm{mod}\, p)$. Thus, in any case, we have the following corollary to Euler's Theorem.

Theorem 7. *(***Fermat's Little Theorem***) Let p be a prime. If a is any integer, then $a^p \equiv a (\mathrm{mod}\, p)$ and $a^{p-1} \equiv 1 (\mathrm{mod}\, p)$ if and only if $p \nmid a$.*

Theorem 7 can be used to show that a number is *not* prime. For example, $2^{32} = 2^2(2^5)^6 = 4(32)^6 \equiv 4(-1)^6 = 4 (\mathrm{mod}\, 33)$ shows that 33 is not prime. However, $2^{10} = (2^5)^2 = (32)^2 \equiv (-1)^2 = 1 \ (\mathrm{mod}\, 11)$ shows that 11 *may* be prime. More evidence is given by the fact that $3^{10} = (3^5)^2 = (243)^2 \equiv (1)^2 = 1 (\mathrm{mod}\, 11)$, but it still doesn't *prove* that 11 is prime.

Corollary 8. *Let p and q be different odd primes, let $m = pq$, and suppose that $r \equiv 1 (\mathrm{mod}\, (p-1))$ and $r \equiv 1 (\mathrm{mod}\, (q-1))$. If a is any integer, then $a^r \equiv a (\mathrm{mod}\, m)$.*

Proof. If $p \nmid a$, then $a^r = a^{k(p-1)+1} = (a^{p-1})^k(a) \equiv (1)^k(a) = a (\mathrm{mod}\, p)$. If $p \mid a$, then $a \equiv 0 \equiv a^r (\mathrm{mod}\, p)$. In either case, $a^r \equiv a (\mathrm{mod}\, p)$ and $p \mid a^r - a$. Similarly, $q \mid a^r - a$. Since both p and q divide $a^r - a$, it follows that $m = pq$ divides $a^r - a$ and hence that $a^r \equiv a (\mathrm{mod}\, m)$. □

5 The RSA Algorithm

Now we are finally able to describe the RSA public key cryptosystem! The RSA algorithm is actually a **cipher**, which means that it works on letters of the alphabet or on the symbols used to write a language rather than on words or meaningful phrases of the language. It really acts on a collection of numbers, so the first job is to get a uniform method of converting the symbols we want to transmit into numbers. One method would be to replace A with 01, B with 02, ..., Z with 26, and communicate only with these 26 letters. Another method would employ the ASCII code. We assume that some such uniformly understood technique has been established throughout the cryptosystem, and will not concern ourselves with it any more. For us, a message will be a number!

Since the RSA algorithm is a cipher, we will use the terms "encipher" and "decipher" that apply to ciphers ("encode" and "decode" are the corresponding terms for codes) rather than the more general terms "encrypt" and "decrypt" that apply to both ciphers and codes, and will make other changes in terminology as appropriate.

Here is what to do in order to set up an RSA cipher. We will discuss how to do it in §6 and §7.

(1) **Secretly** choose two large primes p and q, say each of about 100 digits, with $100 < q/p < 10000$, so that q has 2 to 4 more digits than p. For a small example, let $p = 3$ and $q = 11$.

(2) Let $m = pq$. Note that $\mathbb{Z}_m = \{0, 1, 2, \ldots, m-1\}$ contains q multiples of p, namely $0 \cdot p$, $1 \cdot p$, $2 \cdot p$, ... , $(q-1) \cdot p$, and p multiples of q. These are the only elements of $\mathbb{Z}_m$ which have factors in common with m, and so $\mathbb{Z}_m^*$ contains the remaining $m - p - q + 1 = (p-1)(q-1)$ elements of $\mathbb{Z}_m$ that are relatively prime to m. (The "+1" compensates for the fact that 0 was counted twice, once among the q multiples of p and again among the p multiples of q.) Thus, $\varphi(m) = |\mathbb{Z}_m^*| = m - p - q + 1 = (p-1)(q-1)$. Abbreviate $\varphi(m)$ by φ. In our example, $m = 33$ and $\varphi = 20$.

(3) Choose $e > 10^5$ such that $\gcd(\varphi, e) = 1$ and **secretly** find d such that $ed \equiv 1 (\mathrm{mod}\, \varphi)$. That is, $d \equiv e^{-1} (\mathrm{mod}\, m)$, and $ed = k\varphi + 1$ for some integer k. For our example,

$$\varphi = 20, e = 7, \varphi - 2e = 6, -\varphi + 3e = 1, d = 3.$$

(4) Publish the enciphering key (m, e). Keep the deciphering key (m, d) *secret.*

The security of the RSA cipher is based on the ease of finding the deciphering number d when the factorization of $m = pq$ is known and the difficulty of finding d from m and e when the factorization is not known. This will be discussed at greater length in the next two sections.

How is the RSA cipher used to send a message? Anyone can use the public key (m, e) to encipher the message x in $X = \mathbb{Z}_m$ by raising x to the power e and reducing modulo m to obtain $y = xE \equiv x^e (\mathrm{mod}\, m)$. To decipher the message y, the holder of the secret deciphering key (m, d) raises y to the power d and reduces modulo m to obtain $x = yD = y^d (\mathrm{mod}\, m)$. For our example with $(m, e) = (33, 7)$, the message $x = 17$ is enciphered as $y = 17E = 17^7 \equiv (17)(-16)^6 \equiv (17)(16^2)^3 \equiv (17)((32)(8))^3 \equiv (17)(-8)^3 = (17)(64)(-8) \equiv (17)(-2)(-8) \equiv (-34)(-8) \equiv (-1)(-8) = 8 (\mathrm{mod}\, 33)$. Then $y = 8$ is deciphered with the secret key $(m, d) = (33, 3)$ to obtain $x = y^3 = 8^3 = 8(8^2) = 8(64) \equiv 8(-2) = -16 \equiv 17 (\mathrm{mod}\, 33)$.

Why does this work? It is a result of Corollary (4.5): $yD = (xE)D \equiv (x^e)^d = x^{ed} = x^{k\varphi+1} \equiv x (\mathrm{mod}\, m)$ because $k\varphi + 1 = k(p-1)(q-1) + 1 \equiv 1$ modulo $p - 1$ and modulo $q - 1$. Similarly, $x = yD \equiv y^d (\mathrm{mod}\, m)$ implies $xE = (yD)E \equiv (y^d)^e = y^{de} \equiv y (\mathrm{mod}\, m)$. Thus, E and D are one to one functions from the set $X = Y = \mathbb{Z}_m$ onto itself; that is, they are *permutations* of $\mathbb{Z}_m$. They have the property that for any x and y in $\mathbb{Z}_m$, $xE = y$ if and only if $yD = x$. The **decipherer** (decryptor for a cipher) D is the **inverse function** $D = E^{-1}$ of the **encipherer** (encryptor for a cipher) E, and vice versa, $E = D^{-1}$. Each undoes what the other does!

Let us suppose that Alice and Bob have independently taken the above four steps to set up their RSA ciphers. Alice has published her enciphering key (the encrypting key used with a cipher) (m_A, e_A) and has kept her

deciphering key (m_A, d_A) secret. Alice's set of plaintext and enciphered "messages" are both the same set $X_A = Y_A = \mathbb{Z}_{m_A} = \{0, 1, 2, \ldots, m_A - 1\}$ of all nonnegative integers less than m_A. Alice's encipherer E_A transforms any x in $X_A = \mathbb{Z}_{m_A}$ to $y = xE_A = x^{e_A} (\mathrm{mod}\, m)_A$, which is also in $\mathbb{Z}_{m_A}$. So Bob sends $y = xE_A = x^{e_A} (\mathrm{mod}\, m)_A$ to Alice. When Alice receives y, she applies her decipherer D_A to y to obtain $yD_A = y^{d_A} (\mathrm{mod}\, m)_A = x$.

Now Bob has public enciphering key (m_B, e_B), private deciphering key (m_B, d_B), and message set $X_B = Y_B = \mathbb{Z}_{m_B} = \{0, 1, 2, \ldots, m_B - 1\}$. Assume that $m_A < m_B$. Then $X_A \subsetneqq X_B$. If Bob wants to send a signed message x to Alice, he first applies her public encipherer E_A to obtain $y = xE_A$ in $X_A \subsetneqq X_B$. Since y is also in X_B, he can apply his decipherer D_B to obtain $z = yD_B = xE_AD_B$ in X_B, which he sends to Alice. When Alice receives the message z, she knows it is supposed to have come from Bob and that $X_A \subsetneqq X_B$, so she first applies Bob's publicly known encipherer E_B to obtain $zE_B = y$ in X_B. But *y is also in X_A*, so Alice can then apply her secret decipherer D_A to read the message $zE_BD_A = yD_A = x$. As explained in §**2**, Alice knows the message was originated by Bob, and she can prove that Bob sent the message to an impartial arbiter if Bob later denies it. Bob must be careful to apply E_A first and D_B second, because $y' = xD_B$ might not be in the domain $X_A = \mathbb{Z}_{m_A}$ of E_A and so $y'E_A$ may be undefined. Worse than this, if Bob carelessly calculates $z' = (y')^{e_A} (\mathrm{mod}\, m_A) = ((x^{d_B}) (\mathrm{mod}\, m)_B)^{e_B} (\mathrm{mod}\, m)_A$, then Alice may not be able to recover x from z'. We will see what can happen in the examples below.

In like manner, Alice can send a signed message u to Bob by forming $v = uD_A$ and then $w = vE_B = uD_AE_B$ that Bob can read as $u = wD_BE_A$. Alice also must exercise care in applying D_A and E_B in the correct order.

For example, suppose Alice has enciphering key $(m_A, e_A) = (33, 7)$ and deciphering key $(m_A, d_A) = (33, 3)$, and Bob has keys $(m_B, e_B) = (65, 11)$ and $(m_B, d_B) = (65, 35)$, and Bob wants to send the signed message $x = 18$ to Alice. (The reader can follow the calculations by using the repeated squaring algorithm described in the next section or a computer program like MAPLE.) Bob then calculates $y = 18E_A = (18^7) (\mathrm{mod}\, 33) = 6$ and then $z = 6D_B = (6^{35}) (\mathrm{mod}\, 65) = 11$, which he sends to Alice. Alice then applies Bob's encipherer E_B to $z = 11$ to obtain $y \equiv 11^{11} (\mathrm{mod}\, 65) = 6$, and then applies her decipherer D_A to $y = 6$ to obtain $x = 6D_A \equiv 6^3 (\mathrm{mod}\, 33) = 18$ in order to read the original message $x = 18$.

But what happens if Bob applies his deciphering key first and then his enciphering key? Bob transforms $x = 18$ to $y' = 18D_B \equiv 18^{35} (\mathrm{mod}\, 65) = 47$, and then calculates $z' = 47E_A = 47^7 (\mathrm{mod}\, 33) = 20$, not realizing that $y' = 47$ is not in the domain of E_A. If Alice calculates either $y_1 = z'D_A = 20^3 (\mathrm{mod}\, 33) = 14$ and then $x_1 = y_1E_B = 14^{11} (\mathrm{mod}\, 65) = 14$ or $y_2 = z'E_B = 20^{11} (\mathrm{mod}\, 65) = 15$ and then $x_2 = y_2D_A = 15^3 (\mathrm{mod}\, 33) = 9$, neither $x_1 = z'D_AE_B = 14$ nor $x_2 = z'E_BD_A = 9$ is the message $x = 18$ that Bob intended. *The order of application of operators is important!* When *sending* a signed message the operator (encipherer or decipherer) associated with the

smaller modulus m must be applied first, and when *receiving* a signed message the operator associated with the larger modulus must be applied first.

The reader is invited to construct more examples of signed messages between Alice and Bob. The author was amused to find that Bob could have successfully sent the sample message $x = 17$ to Alice as a signed message with operators reversed because $17D_B = 17^{35} (\mathrm{mod}\, 65) = 23$ turned out to be in X_A.

6 Algorithms and Time Estimates

In order to *set up* an RSA cipher algorithm, one must first find two large primes p and q, find a number e relatively prime to $\varphi = (p-1)(q-1)$, and then find the inverse d of e modulo φ. Then, in order to *use* the algorithm, one must calculate the residues of x^e and y^d modulo $m = pq$, and all of this has to be "easy" for numbers d, e, φ, m, x, y with up to 200 digits each! But it has to be "hard" to factor m without a prior knowledge of p, q, φ, or d. In this section and in the next we explain why these tasks are "easy" or "hard". Much of this material in this section is covered in greater detail in sections 1 and 3 of Chapter I of [Ko].

The ease or difficulty of performing an arithmetic procedure can be measured by the number of **bit operations** needed to carry it out. A bit operation is the basic computer step used to calculate a single bit in the addition or subtraction of two numbers in binary notation. Thus the addition or subtraction of two k-bit integers requires k bit operations. Now a positive integer $a = a_0 + a_1 b + \cdots + a_{k-1} b^{k-1}$ with $0 \leq a_j < b$ and $a_{k-1} \neq 0$ has $k - 1 \leq \log_b a < k$ and hence has $k = 1 + \lfloor \log_b a \rfloor$ base b "digits". (Here $\lfloor t \rfloor$ denotes the greatest integer less than or equal to t. $\lfloor t \rfloor$ is called the **floor** of the real number t, and it satisfies $\lfloor t \rfloor \leq t < \lfloor t \rfloor + 1$. For example, $\lfloor 3.14 \rfloor = 3 = \lfloor 3 \rfloor$ and $\lfloor -3.2 \rfloor = -4 = \lfloor -4 \rfloor$. Most computer languages and calculators use $INT(t)$ for $\lfloor t \rfloor$.) Thus the positive integer a has $1 + \lfloor \log_2 a \rfloor$, or about $\log_2 a$, bits. For integers a and b we write

$$T(a \pm b) = O(\log_2 a) \text{ when } 0 < b \leq a \tag{6}$$

to indicate that addition (or subtraction) takes time proportional to the number of bits in the largest addend.

The "big O-notation" is defined as follows. If f and g are functions of n variables $x_1, x_2, \cdots, x_n$, we say that f is bounded by g, and write $f(x_1, x_2, \cdots, x_n) = O(g(x_1, x_2, \cdots, x_n))$ or $f = O(g)$, if there are constants B and $C > 0$ such that $0 < f(x_1, x_2, \cdots, x_n) < Cg(x_1, x_2, \cdots, x_n)$ whenever all of the $x_j > B$. The reader may find it helpful to simply interpret the big "O" as a fixed but unknown positive constant.

Consider the product $27 \times 11 = 297$ in binary (base 2) notation.

$$\begin{array}{r} 1\,1\,0\,1\,1 \\ 1\,0\,1\,1 \\ \hline 1\,1\,0\,1\,1 \\ 1\,1\,0\,1\,1 \\ 1\,1\,0\,1\,1 \\ \hline 1\,0\,0\,1\,0\,1\,0\,0\,1 \end{array}$$

From this example we see that if a has k bits and b has j bits with $j \leq k$, we can find the product by writing one copy of the multiplicand a with its unit bit aligned beneath each 1 in the multiplier b and then adding these staggered copies of a pairwise to form the product. There are no more than j additions, each taking no more than $j + k \leq 2k$ bit operations, and so the whole process requires no more than $j(j + k) \leq 2jk = O(\log_2 b \log_2 a)$ bit operations. This yields

$$T(a \times b) = O((\log_2 a)(\log_2 b)) = O((\log_2 a)^2) \quad \text{when} \quad 0 < b \leq a. \tag{7}$$

A similar analysis of the grade school long division algorithm leads to the same result for division of the positive integer a by the positive integer d to obtain the quotient q and the remainder r satisfying the division algorithm (2). We indicate the time required by

$$T(q, r : a = dq + r) = O((\log_2 a)(\log_2 d)) = O((\log_2 a)^2) \quad \text{when} \quad 0 < d \leq a. \tag{8}$$

Note that if we want to find the residue $\overline{a}$ modulo m of an integer a we need only use the division algorithm to find q and $\overline{a}$ such that $a = mq + \overline{a}$ with $0 \leq \overline{a} < m$, so it follows that the time needed for the procedure is

$$T(\overline{a} : a \equiv \overline{a} (\mathrm{mod}\, m) \text{ and } 0 \leq \overline{a} < m) = O((\log_2 a)(\log_2 m)). \tag{9}$$

We should remark that we are obtaining crude upper bounds for the times required to perform various calculations which can be lowered considerably by carefully examining more sophisticated algorithms presently in use. Our goal is just to get a rough idea of the difficulty involved in the RSA calculations.

We want to extend the Euclidean algorithm (4) to one that will find not only $d = \gcd(a, b)$, but also find x and y so that $d = \gcd(a, b) = xa + yb$. We use vector notation $\overline{v} = \langle v_1, v_2, v_3 \rangle$ and write the algorithm in pseudo computer code, using the notation "$\overline{u} \leftarrow \overline{v}$" to mean "replace $\overline{u}$ by the value of $\overline{v}$".

Theorem 9. *(***Extended Euclidean Algorithm***) Algorithm for* $d = \gcd(a, b) = xa + yb$ *for* $0 \leq b < a$.

(a) $\overline{u} \leftarrow \langle a, 1, 0 \rangle$

(b) $\overline{v} \leftarrow \langle b, 0, 1\rangle$
(c) *do* $\overline{w} \leftarrow \overline{u} - \lfloor u_1/v_1 \rfloor \overline{v}$, $\overline{u} \leftarrow \overline{v}$, $\overline{v} \leftarrow \overline{w}$ *while* $v_1 \neq 0$
(e) $d \leftarrow u_1$, $x \leftarrow u_2$, $y \leftarrow u_3$
(f) *return* d, x, y.

We repeat example (3.6) in matrix form, with the sequence of vectors $\overline{u}$ and $\overline{v}$ written as rows of the matrix. On the right we write the example as we might do it "by hand". Both forms illustrate that $\gcd(54321, 12345) = 3 = (-822)(54321) + (3617)(12345)$.

$$
(6.6)\quad \left[\begin{array}{rrr}
54321 & 1 & 0 \\
12345 & 0 & 1 \\
4941 & 1 & -4 \\
2463 & -2 & 9 \\
15 & 5 & -22 \\
3 & -822 & 3617 \\
0 & 4115 & -18107
\end{array}\right]
\quad
\begin{array}{rrr}
a & & = 54321 \\
 & b & = 12345 \\
a - & 4b & = 4941 \\
-2a + & 9b & = 2463 \\
5a - & 22b & = 15 \\
-822a + & 3617b & = 3 \\
4115a - & 18107b & = 0
\end{array}
$$

This process really amounts to the row reduction of the matrix $\begin{bmatrix} a & 1 & 0 \\ b & 0 & 1 \end{bmatrix}$ to $\begin{bmatrix} d & x & y \\ 0 & u & v \end{bmatrix}$ with $d = \gcd(a, b) = xa + yb$ and $0 = ua + vb$. At the k^{th} step, starting at $k = 0$ with $r_{-1} = a$, $r_0 = b$, $x_{-1} = 1$, $x_0 = 0$, $y_{-1} = 0$, and $y_0 = 1$, we have

$$
\begin{bmatrix} r_k & x_k & y_k \\ r_{k+1} & x_{k+1} & y_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_{k+1} \end{bmatrix} \begin{bmatrix} r_{k-1} & x_{k-1} & y_{k-1} \\ r_k & x_k & y_k \end{bmatrix},
$$

where the r_k and $q_{k+1} = \lfloor r_{k-1}/r_k \rfloor$ are the remainders and quotients displayed in (4). Note that at every stage we have $r_k = x_k a + y_k b$. This technique has been discussed several times (See [Bl] and [MW].) in the literature.

Consider rows (k) and (k+1) of (4):

(k) $r_{k-1} = r_k q_{k+1} + r_{k+1}$,
(k+1) $r_k = r_{k+1} q_{k+2} + r_{k+2}$.

If $r_{k+1} \leq \frac{1}{2} r_k$ it follows from the division algorithm that $0 \leq r_{k+2} < r_{k+1} < \frac{1}{2} r_k$, and if $r_{k+1} > \frac{1}{2} r_k$, then $q_{k+2} = 1$ and $r_{k+2} = r_k - r_{k+1} < \frac{1}{2} r_k$. Eventually $r_{2k} < \frac{1}{2^k} a < 1$, so the algorithm must terminate in about $n = 2 \log_2 a$ steps. The calculation involved for each step occurs in line (c) of (9) and involves one division and three multiplications of numbers no larger than a, and so requires no more than $4(\log_2 a)^2$ bit operations, by (8) and (7). The whole algorithm can be accomplished with no more than $8(\log_2 a)^3$ bit operations. This proves

$$
T(d, x, y : d = \gcd(a, b) = xa + yb) = O((\log_2 a)^3) \quad \text{when} \quad 0 < b < a. \tag{10}
$$

Now we turn our attention to the problem of evaluating $x^e \pmod m$. For this purpose we consider the following construction.

Theorem 10. *(***Mod Power algorithm***) To evaluate $x^e \pmod m$, perform the following algorithm:*

(a) $E \leftarrow e$, $B \leftarrow x$, $P \leftarrow 1$,
(b) do until $E = 0$
(c) if E *even*
(d) $E \leftarrow E/2$, $B \leftarrow B \cdot B \pmod m$
(e) else
(f) $E \leftarrow E - 1$, $P \leftarrow P \cdot B \pmod m$
(g) end if
(h) return P.

Proof. It suffices to show that $P \cdot B^E \equiv x^e \pmod m$ at every step. This is clear in line (a). If true before line (d) is executed, it holds afterward, because $P \cdot B^E = P \cdot (B \cdot B)^{E/2}$. And if true before line (f) is executed, it holds afterward, because $P \cdot B^E = (P \cdot B) \cdot B^{E-1}$. Therefore $P \equiv x^e \pmod m$ when $E = 0$. □

When E is in binary form, line (c) is performed by checking whether the final bit is 0 or not. If so, $E \leftarrow E/2$ is performed by dropping final 0, and if not, $E \leftarrow E-1$ is performed by changing the final 1 to a 0, so these operations take negligible time. Since B and P are both less than m, we see that each of the products $B \cdot B$ and $P \cdot B$ takes no more than $O((\log_2 m)^2)$ bit operations, by (6.2). Then replacing a by m^2 in (9) shows that each of the operations $B \leftarrow B \cdot B \pmod m$ and $P \leftarrow P \cdot B \pmod m$ takes $O((\log_2 m)^2)$ bit operations. Adding these two $O((\log_2 m)^2)$ gives $O(2(\log_2 m)^2) = O((\log_2 m)^2)$ for the execution of either line (d) or (f). After each execution of line (f), E becomes even, so each execution of (f) must be followed by an execution of (d), thereby halving E. Therefore, there can be at most $2 \log_2 e$ passes through the loop (b). Hence the algorithm requires no more than $O((\log_2 e)(\log_2 m)^2)$ bit operations.

$$T(x^e \pmod m) = O((\log_2 e)(\log_2 m)^2). \tag{11}$$

We end this discussion of $x^e \pmod m$ with a numerical example in which we use the algorithm (10) to evaluate $18^7 \pmod{33}$, $47^7 \pmod{33}$, $6^{35} \pmod{65}$, and $18^{35} \pmod{65}$. The reader should refer back to (10) to be sure he understands the changes in B, E, and P.

(mod 33)		
B	E	P
18	7	1
18	6	18
27	3	18
27	2	24
3	1	24
3	0	6

$18^7 (\mathrm{mod}\, 33) = 6$

(mod 33)		
B	E	P
47	7	1
47	6	47
31	3	47
31	2	5
4	1	5
4	0	20

$47^7 (\mathrm{mod}\, 33) = 20$

(mod 65)		
B	E	P
6	35	1
6	34	6
36	17	6
36	16	21
61	8	21
16	4	21
61	2	21
16	1	21
16	0	11

$6^{35} (\mathrm{mod}\, 65) = 11$

(mod 65)		
B	E	P
18	35	1
18	34	18
64	17	18
64	16	47
1	8	47
1	4	47
1	2	47
1	1	47
1	0	47

$18^{35} (\mathrm{mod}\, 65) = 47$

We conclude this section with the following definition which is crucial to the study of time estimates for algorithms.

Definition 11. An algorithm to perform a computation involving n integers $x_1, x_2, \cdots, x_n$ of $k_1, k_2, \cdots, k_n$ bits, respectively, is a **polynomial time algorithm** if there are positive integers $d_1, d_2, \cdots, d_n$ such that the number of bit operations required to perform the algorithm is $O(k_1^{d_1} k_2^{d_2} \cdots k_n^{d_n})$. In this case, we say it is of **degree** d_j **in** k_j (or $\log_2 x_j$) and that it is of **total degree** $d_1 + d_2 + \cdots + d_n$.

All of the algorithms presented in this section are of polynomial time. Basically, polynomial time algorithms are considered to be easy, and non-polynomial time algorithms are considered to be hard!

7 Implementation

The first job in implementing an RSA cipher is to secretly find two primes p and q of about $98 - 99$ and $101 - 102$ digits, respectively, so that $m = pq$ has

about 200 digits. For greater security we can choose larger primes. In order to make them hard for our antagonists to discover, we should *randomly* select the primes. One way of doing this is to use a random number generator to pick an integer with the appropriate number of digits, add one if it is even, and test the resulting integer n for primality.

But how can we tell that an odd integer n is prime? We could try the "dumb test": Try to divide n by every positive integer $d \leq \sqrt{n}$. This could take about 10^{50} trial divisions, each taking an average of $(\log_2 10^{50})^2 \approx 27,000$ bit operations, for a total of about 3×10^{54} bit operations. If our computer can do 10^9 bit operations a second, this could take 3×10^{45} seconds, or about 8×10^{38} years.

Fortunately, there are better ways! We know by Fermat's Little Theorem 7 that if n is prime and $1 < b < n$, then

$$b^{n-1} \equiv 1 (\mathrm{mod}\, n). \tag{12}$$

Hence, if (12) does not hold for some b satisfying $1 < b < n$, then we *know* that n is composite, and we call b a **witness** to the fact that n is not prime. On the other hand, n could be an odd composite number and (12) could still hold for some b satisfying $1 < b < n$, but this is not likely.

Definition 12. If n is an odd composite number and (12) holds for some b satisfying $1 < b < n$, then we say that n is a **pseudoprime to the base** b.

For example, 91 is a pseudoprime to the base 3 and to the base 10, since $3^{90} \equiv 10^{90} \equiv 1 (\mathrm{mod}\, 91)$. However, both 2 and 5 are witnesses to the compositeness of 91, since $2^{90} \equiv 5^{90} \equiv 64 (\mathrm{mod}\, 91)$. These congruences are fairly easy to calculate once you realize that $3^6 \equiv 1 (\mathrm{mod}\, 91)$ because $3^6 \equiv 1 (\mathrm{mod}\, 91)$ implies that $10^6 = (10^2)^3 \equiv 9^3 = 3^6 \equiv 1 (\mathrm{mod}\, 91)$ and $64 \equiv -27 (\mathrm{mod}\, 91)$ implies that $64^2 \equiv 27^2 = 3^6 \equiv 1 (\mathrm{mod}\, 91)$, so $2^{90} = (2^6)^{15} = 64^{15} = 64(64^2)^7 \equiv 64(1)^7 = 64 (\mathrm{mod}\, 91)$ and $5^6 \equiv (2^6)^{-1} = 64^{-1} \equiv 64 (\mathrm{mod}\, 91)$ because $(2^6)(5^6) = 10^6 \equiv 1 (\mathrm{mod}\, 91)$.

Unfortunately, there are odd positive composites n, called **Carmichael numbers** , such that (12) holds whenever $\gcd(n, b) = 1$. As recently as 1994, Alford, Granville, and Pomerance showed in [AGP] that there are infinitely many Carmichael numbers. The smallest Carmichael number is $561 = 3 \times 11 \times 17$. Note that (12) holds for $b = 2, 4, 5, 7, 8, 10$, and 13, but of course it can't hold for $b = 3, 6, 9, 11, 12, 15$, or 17, or any other b with $\gcd(561, b) > 1$, since $b^k \equiv 1 (\mathrm{mod}\, n)$ implies $gcd(n, b) = 1$.

We remarked that $2^{560} \equiv 1 (\mathrm{mod}\, 561)$. Note that $560 = 35 \times 2^4$ and $2^{35} \equiv 263 (\mathrm{mod}\, 561)$, $2^{35 \times 2} \equiv 263^2 \equiv 166 (\mathrm{mod}\, 561)$, $2^{35 \times 2^2} \equiv 166^2 \equiv 67 (\mathrm{mod}\, 561)$, $2^{35 \times 2^3} \equiv 67^2 \equiv 1 (\mathrm{mod}\, 561)$. This last congruence implies that $561 \mid 67^2 - 1 = (67 - 1)(67 + 1) = 66 \times 68$. But clearly $561 \nmid 66$ and $561 \nmid 68$, and so both $\gcd(561, 66) = 33$ and $\gcd(561, 68) = 17$ are nonunit factors of $561 = 33 \times 17$. Several useful lessons can be learned from this example.

Lemma 13. *If a and n are positive integers with n odd and $1 < a < n-1$ such that $a^2 \equiv 1 (\mathrm{mod}\, n)$, then $\gcd(n, a-1) > 1$, $\gcd(n, a+1) > 1$, and $n = \gcd(n, a-1)\gcd(n, a+1)$ is composite.*

Proof. By hypothesis, $n \mid a^2 - 1 = (a-1)(a+1)$. Note that we have $3 < a < n-1$, since $a = 2$ and $a = 3$ yield contradictions. But $a \neq \pm 1 (\mathrm{mod}\, n)$ implies $n \nmid a \mp 1$ implies $\gcd(n, a-1) > 1$ and $\gcd(n, a+1) > 1$. The fact that $n \mid (a-1)(a+1)$ implies that $n \mid \gcd(n, a-1)\gcd(n, a+1)$. Since $\gcd(a-1, a+1) = \gcd(a-1, 2) = 1$ or 2 and n is odd, $\gcd(n, a-1)$ and $\gcd(n, a+1)$ are relatively prime, and they both divide n, and so their product divides n. That is, $n \mid \gcd(n, a-1)\gcd(n, a+1) \mid n$, so $n = \gcd(n, a-1)\gcd(n, a+1)$. □

Definition 14. Suppose that n is an odd composite number and $n-1 = 2^s t$ with t odd. If $b \in \mathbb{Z}_n^*$ satisfies either $b^t \equiv 1 (\mathrm{mod}\, n)$ or $b^{2^r t} \equiv -1 (\mathrm{mod}\, n)$ for some r such that $0 \leq r < s$, then n is called a **strong pseudoprime to the base** b.

The next theorem is a quotation of **Proposition V.1.7** from [Ko], page 130. The reader is referred to [Ko] for the proof.

Theorem 15. *If n is an odd composite integer, then n is a strong pseudoprime to the base b for at most 25% of all $0 < b < n$.*

This suggests a probabilistic test for the primality of an odd integer n, called the **Miller-Rabin primality test**. First, write $n - 1 = 2^s t$ with t odd. This takes negligible time if n is written in binary form: s is just the number of trailing zeros in $n-1$ and t is the number left when these zeros are dropped. Randomly select an integer b satisfying $1 < b < n$. Use the Mod Power algorithm (10) to evaluate $c \equiv b^t (\mathrm{mod}\, n)$ in time $O((\log_2 n)^3)$. If either $c = 1$ or $c^{2^r} \equiv -1 (\mathrm{mod}\, n)$ for some r with $0 \leq r < s$, we say that n passes the test for base b. In this case, n is either a prime or is a strong pseudoprime to the base b. When $c \neq 1$ we must square and reduce the result modulo n, doing this r times with $0 \leq r < s < \log_2 n$, which can be done in time $rO((\log_2 n)^2) = O((\log_2 n)^3)$,. by (6.2) and (9). Suppose that n passes the test for k randomly chosen values of b with $1 < b < n$. It follows from Theorem (7.4) that the probability that n is composite is $\leq 1/4^k$.

If n passes the test for 50 or more values of b, we might call it an **industrial grade prime**. For a 100 digit odd number n this could take on the order of

$$50(\log_2 n)^3 \approx 50(333)^3 \approx 2 \times 10^9$$

bit operations, or 2 seconds on the very fast computer hypothesized at the beginning of this section. If we assume, perhaps more realistically, that our computer can perform 10 million bit operations per second, then it takes on the order of 3 or 4 minutes.

For each positive integer n, let $\pi(n)$ be the number of primes less than or equal to n. The **Prime Number Theorem** says thatlim$(\frac{\pi(n)\log n}{n}) = 1$ as $n \longrightarrow \infty$. Thus $\pi(n) \approx \frac{n}{\log n}$ for large values of n and the frequency of primes near a large value of n is about $\frac{1}{\log n}$, so one would expect to test $O(\log n)$ numbers in order to find a prime bigger than n. Putting all of this together gives the following result.

Lemma 16. *An industrial grade prime $p \geq n$ can be found with $O((\log_2 n)^4)$ bit operations.*

The purist may be disappointed in not having a *definitive* polynomial time primality test that actually *proves* that a number is prime. Although not polynomial time, there is a definitive primality test which in practice can prove primality of hundred digit numbers in a matter of seconds. It is described in [CL].

Great: now we can secretly produce two large primes p and q in order to set up an RSA cryptosystem. Let us assume that $p < q$. The calculation of $m = pq$ takes only $O((\log_2 p)(\log_2 q)) = O((\log_2 q)^2)$ bit operations by (6.2), and then $\varphi = \varphi(m) = (p-1)(q-1) = m - p - q + 1$ is a bargain at $O(\log_2 q)$ by (6.2)! Finally, we want to find numbers e and d such that $ed \equiv 1 (\mathrm{mod}\, m)$. One method is to randomly choose fairly large integer values of e and calculate the gcds until $\gcd(\varphi, e) = 1 = x\varphi + de$. Since we expect to find a prime close to φ in $\log \varphi$ tries, we should need no more to find a number relatively prime to φ. Combining this with (6.7), finding e and d can be done in $O((\log_2 \varphi)^4)$ bit operations. Another way to find an e relatively prime to φ is just to locate a prime $e > q$, which again is an $O((\log_2 \varphi)^4)$ process.

Therefore we can set up an RSA cryptosystem with modulus m in polynomial time, specifically, with $O((\log_2 m)^4)$ bit operations. But what about using it to communicate? It follows from (11) that messages can be enciphered in time $T(x^e (\mathrm{mod}\, m)) = O((\log_2 e)(\log_2 m)^2)$. Likewise, the deciphering time is $T(x^d (\mathrm{mod}\, m)) = O((\log_2 d)(\log_2 m)^2)$. This concludes the "easy" part: We have seen that it is "easy" to set up and use an RSA cryptosystem.

8 Security

Factoring products of large primes is believed to be very difficult. This belief arises from the fact that people have been trying hard to accomplish such factorizations efficiently for thousands of years without much success. The security of the RSA cryptosystem is based on the belief that breaking the cipher is equivalent to factoring the modulus m given in the public key (m, e).

Of course, if one could find a factorization of the modulus $m = pq$ used for an RSA system with public key (m, e), one would be able to find $\varphi = \varphi(m) = m - p - q + 1$ and use the extended Euclidean algorithm to find $\gcd(\varphi, e) = x\varphi + de = 1$ and d, and thereby obtain the secret deciphering

key (m, d). Conversely, knowing φ is sufficient to factor m. We assume that $p < q$.

$$A = p + q = m + 1 - \varphi,$$
$$A^2 = (p+q)^2 = p^2 + 2pq + q^2 = p^2 + 2m + q^2,$$
$$A^2 - 4m = p^2 - 2m + q^2 = p^2 - 2pq + q^2 = (p-q)^2,$$
$$B = q - p = \sqrt{A^2 - 4m},$$
$$p = (A - B)/2 \text{ and } q = (A + B)/2.$$

Hence, knowing φ is equivalent to being able to factor m.

But we still don't know that we can't break the RSA by some method that does not lead to a factorization of the enciphering modulus m. A complete solution to the cipher would mean being able to recover *every* x in $\mathbb{Z}_m$ from its encipherment $y \equiv x^e (\mathrm{mod}\, m)$. Could this be done without a knowledge of the unique $d \equiv e^{-1} (\mathrm{mod}\, \varphi)$? The answer to this question is yes, as we can see from our example with enciphering key $(m, e) = (33, 7)$. Recall that the deciphering key was $(m, d) = (33, 3)$. We will see that $(m, d') = (33, 13)$ works just as well! For if $y \equiv x^7 (\mathrm{mod}\, 33)$, then $y^{13} \equiv (x^7)^{13} = x^{91} \equiv x (\mathrm{mod}\, 33)$ by Corollary 8 because $91 \equiv 1 (\mathrm{mod}\, 2)$ and $91 \equiv 1 (\mathrm{mod}\, 10)$.

Okay, suppose that one has a number d' such that $x^{ed'} \equiv x (\mathrm{mod}\, m)$ for every x in $\mathbb{Z}_m$. That means that one has a number $b = ed' - 1$ such that $x^b \equiv 1 (\mathrm{mod}\, m)$ for every x in $\mathbb{Z}_m^*$. b must be even because $(-1)^b \equiv 1 (\mathrm{mod}\, m)$. Then it turns out that there are positive integers r and a (See [Ko], p. 94.) such that $b = 2^r a$, $x^{2a} \equiv 1 (\mathrm{mod}\, m)$ for every x in $\mathbb{Z}_m^*$, but there is some x in $\mathbb{Z}_m^*$ such that $x^a \neq 1 (\mathrm{mod}\, m)$. In this case, $x^a \neq 1 (\mathrm{mod}\, m)$ for at least 50% of the values of x in $\mathbb{Z}_m^*$, and so choosing random elements of $\mathbb{Z}_m^*$ should lead to such an x fairly quickly. And for such an a, there are at least 50% of the x in $\mathbb{Z}_m^*$ for which $x^a - 1$ is divisible by one of the primes p or q, but not both. Then $\gcd(m, x^a - 1)$ is one of the two primes p or q. This gives a factorization of m.

Although we have certainly not shown that breaking the RSA algorithm is equivalent to factoring the modulus m in the public key (m, e), there is certainly a close relationship. And it is clear that the cipher is weak if m is easy to factor. One problem occurs if p is close to q. We can write $m = pq = (t + s)(t - s) = t^2 - s^2$ with $q = t + s$, $p = t - s$, so $t = (q + p)/2$ and $s = (q - p)/2$, where we make our usual assumption that $p < q$. If p is close to q, then $s^2 = t^2 - m$ is a small perfect square. So we attempt to factor m by taking $t = \lfloor \sqrt{m} \rfloor + k$ for small k. Consider an example from [Ko], p.144: Factor $m = 200819$. $\lfloor \sqrt{200819} \rfloor = 448$, so we try $t = 448 + k$. For $k = 1$, $t^2 - m = 449^2 - 200819 = 782$ is not a perfect square. For $k = 2$, $t^2 - m = 450^2 - 200819 = 1681 = 41^2$, so $s = 41$ and $t = 450$ gives the factorization $p = t - s = 409$ and $q = t + s = 491$. This method, called **Fermat factorization**, and generalizations such as the *factor base method* and the *quadratic sieve method* are especially effective when p and q are close together. There are other methods which are effective when $p - 1$ and $q - 1$ have many small factors.

It behooves the user of an RSA public key cipher to avoid the situations mentioned above, as well as any others which may arise because of new factoring methods. However, no known factorization methods are polynomial time algorithms, so it seems likely that the cipher can stay ahead of the factorizations by choosing larger primes. A number of additional precautions should be taken when implementing an RSA cryptosystem in order to make it secure. But if these precautions are taken, it seems that the system is here to stay! These questions are more fully addressed in an article [Bo] "Twenty Years of Attacks on the RSA Cryprosystem" by Dan Boneh in which he concludes that there have been some insightful attacks, but no devastating attack has been found, and that with proper implementation the system can be trusted to be secure.

References

[AGP] W. Alford, A. Granville, and C. Pomerance, There are infinitely many Carmichael numbers, *Annals of Math., Vol. 139 (1994), 703-722.*

[Bl] W. A. Blankinship, A new version of the Euclidean algorithm, *The American Mathematical Monthly,* September 1963, 742-745.

[Bo] Dan Boneh, Twenty years of attacks on the RSA cryptosystem, *Notices of the American Math. Soc.,* Vol. 46, No. 2 (February 1999), 203-213.

[CL] H. Cohen and H. W. Lenstra, Jr., Primality testing and Jacobi sums, *Math. Comp. Vol. 42 (1984), 297-330.*

[DH] Whitfield Diffie and Martin E. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory,* Vol. IT-22, No. 6, Nov. 1976; or reprinted in *Secure Communications and Assymmetric Cryptosystems,* AAAS Selected Synposium 69, Westview Press, Boulder, CO, 1982.

[G] Martin Gardner, Mathematical Games, *Scientific American,* August 1977.

[H] Martin E. Hellman, The Mathematics of Public-Key Cryptography, *Scientific American*, August 1979, 146-157.

[K] David Kahn, **The Codebreakers**, MacMillan, New York, 1967.

[MW] Richard Maruszewski and William Wardlaw, A note on the Euclidean algorithm, *The AMATYC Journal, Vol. 17 No. 1, fall 1995, 29-32.*

[Ko] Neal Koblitz, **A Course in Number Theory and Cryptography**, 2ed, Springer, New York, 1994.

[RSA] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, A method for obtaining digital signatures and public key signatures, *Communications of the ACM*, Vol. 21, No. 2, Feb. 1978; or reprinted in *Secure Communications and Assymmetric Cryptosystems,* AAAS Selected Synposium 69, Westview Press, Boulder, CO, 1982.