Notes on Cryptography

Charles Blair
Business Administration Dept.
University of Illinois
c-blair@uiuc.edu

Contents

1 Traditional Encryption Systems **3**

2 Introduction to Number Theory **7**
2.1 Congruences 7
2.2 The Greatest Common Divisor 7
2.3 Powers modulo a prime 9
2.4 Primitive roots 10
2.5 Proof of Theorem 5 11

3 Encryption techniques based on powers and congruences **14**
3.1 The Diffie-Hellman key exchange procedure 14
3.2 The Rivest-Shamir-Adleman public key system 14
3.3 A public key system as hard as factoring 15
3.3.1 Computing square roots modulo a prime 17

4 Subset-Sum (Knapsack) problems and their uses **18**
4.1 Subset-sum problems are hard 18
4.2 A proposed public-key system based on subset-sum 19
4.3 Breaking Knapsack Cryptosystems 21
4.4 Other uses of the subset-sum problem 23
4.4.1 Computer passwords 23
4.4.2 Message verification 23

5 Subset-Sum Problems and NP-Completeness **24**
5.1 The Satisfiability Problem 24
5.2 Converting (SP) to (SSP) 25
5.3 Converting (SSP) to (SP) 26
5.4 Cook's Theorem 28
5.5 Terminology 29

6 A probabilistic test for primality **29**
6.1 Analysis of the Rabin test 30

7 Probabilistic Encryption **32**
7.1 The Goldwasser-Micali encryption system 32
7.2 Weak laws of large numbers 34
7.3 The magic of sampling 36
7.4 Determining algorithm performance by sampling 37
7.5 Two versions of QRA 37
7.6 Knowing a pseudo-square does not help much 38
7.7 The inability to distinguish two plaintexts 38
7.8 Semantic Security 39
7.9 How to play poker over the telephone 39

8 Pseudo-random number generators **42**
8.1 The Quadratic Generator 42
8.2 The Next Bit Theorem 43
8.3 The Efficient Test Theorem 44
8.3.1 A consequence involving symmetry 47

9 Further results on pseudo-random generators **47**
9.1 Hard-Core Predicates and Pseudo-Random Numbers 48
9.2 Construction of hard-core predicates 49
9.3 A recent pseudo-random number generator 52

1 Traditional Encryption Systems

An encryption system is a procedure which takes the original message (*plaintext*) and a small piece of information arranged in advance between sender and receiver (the *key*) and creates an encoded version of the message (the *ciphertext*).

When we are considering the quality of an encryption system, we assume the person trying to decode the message knows what the general procedure is and is looking at the ciphertext— the only thing he does not have is the key. We also assume the person sending messages does not spend time trying to contrive a difficult-to-read message by using unusual words or letter frequencies— the sender is counting on the system to provide all the needed security.

Usually one assumes the person trying to break the code is only working with the ciphertext. However, there are situations in which both plaintext and ciphertext of a previously encoded message are available. For example, I often keep encrypted versions of examinations on a mainframe computer, only decoding them just before having them printed, and deleting the plaintext file immediately afterward. If a student was able to look at my files, he could keep a copy of the encoded test and compare this with the test he took. As we will see, this may be very useful in decoding future tests.

[One countermeasure against this type of *known-plaintext attack* is to continually change keys, assuming an encryption using one key is not helpful on a different key. It can become difficult to keep track of the different keys in use, especially if they are long.]

A more demanding standard is that a code may be safe against a *chosen-plaintext attack*. We are imagining that the encryption is done by a machine, and that unauthorized persons may have access to the machine (although we assume they are only using it in the normal way, not allowed to take it apart).

Example 1: Simple substitution

This is the simple letter-for-letter method found in Poe's "The Gold Bug" and many other stories. The key is a rearrangement of the 26 letters:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
actqgwrzdevfbhinsymujxplok
```

Using this key, the plaintext:

```
THE SECURITY OF THE RSA ENCODING SCHEME RELIES ON THE
FACT THAT NOBODY HAS BEEN ABLE TO DISCOVER HOW TO TAKE
CUBE ROOTS MOD N WITHOUT KNOWING NS FACTORS
```

becomes the ciphertext:

```
UZG MGTJYDUO IW UZG YMA GHTIQDHR MTZGBG YGFDGM IH UZG
WATU UZAU HICIQO ZAM CGGH ACFG UI QDMTIXGY ZIP UI UAVG
TJCG YIIUM BIQ H PDUZIJU VHIPDHR HM WATUIYM
```

The messages can be made harder to decode (but also harder to read!) by leaving out the spaces between words.

Most messages can be decoded by looking for frequently occuring pairs of letters (`TH` and `HE` are by far the most common), using these to identify a few letters to begin, and filling in the remaining letters one at a time ("The Gold Bug" gives a good description, as do many books).

In a known-plaintext situation, the whole code is obtained almost immediately. However, in our example, the letters `J`, `P`, and others do not occur in the plaintext, so we could not tell how they are encoded. If we were allowed a chosen plaintext, we would use all the letters to get the entire key.

Example 2: The Vigenère cipher and one-time pads

This cipher works by replacing each letter by another letter a specified number of positions further in the alphabet. For example `J` is 5 positions further than `E`. `D` is 5 positions after `Y`. (`Y,Z,A,B,C,D`) The key is a sequence of shift amounts. If the sequence is of length 10, the 1st, 11th, 21st, ... letters of the plaintext are processed using the first member of the key. The second member of the key processes plaintext letters 2, 12, 22, ... and so forth. If we omit spaces from the plaintext on page 4 and use the key-sequence:

3 1 7 23 10 5 19 14 19 24

we obtain

```
WILPOHNFBR BPMQRJKGTC QDVASSZGVF
HNLOOQBSLM QUOBPFVHMF DUULLTWMAY VCLBXFUZXR
REPPMTOSKF RXALDFDSVS EFYLYYLAHB QXPQRTNHDL
RXPKQSLTTA WPYP
```

(We have divided the ciphertext into groups of ten letters for convenience. The division into lines is arbitrary.)

This type of cipher was considered secure around the year 1600, but is not really difficult. Suppose we guess that the first letter is `T`. This implies the eleventh letter is `Y`, the 21st letter is `N`, etc. Now look at the two-letter combinations that occur from different possiblilities for the second letter:

```
TI YP ND EN NU AU SC OE OX BF NX OX TP   (no shift of 2nd letter)
TJ YQ NE EO NV AV SD OF OY BG NY OY TQ
TK YR NF EP NW AW SE OG OZ BH NZ OZ TR
TL YS NG EQ NX AX SF OH OA BI NA OA TS
   (skipping over some in the middle)
TF YM NA EK NR AR SZ OB OU BE NU OU TM
TG YN NB EL NS AS SA OC OV BD NV OV TN
TH YO NC EM NT AT SB OD OW BE NW OW TO
```

The last line is the "right answer." Although it shows several bad combinations (`NC NT SB NW`), mostly caused by the last letter of one word being adjacent to the first letter of the next word, it looks better than the other possible rows. Once the second letter has been identified, the same approach can be used to get the third letter. This approach is easily automated using a table of digrams.

It is necessary to know the first letter and the length of the key-sequence. If we assume the length is not too large, a program can just try all possibilities, eventually choosing the plaintext which looks best.[1]

One-time pads

A long key-sequence makes this approach more difficult, since we have fewer rows. The extreme case is that in which the key-sequence is as long as the plaintext itself. This leads to a theoretically *unbreakable* cipher. For any possible plaintext, there is a key for which the given ciphertext comes from that plaintext.

This type of cipher has reportedly been used by spies, who were furnished with notebooks containing page after page of randomly generated

[1] Mike Mendelson, a student in this course in 1989, wrote a program to implement this algorithm. Another method would choose the shift amount for each member of the cycle which gives the best letter frequency.

key-sequence. Notice that it is essential that each key-sequence be used only once (hence the name of the system). Otherwise the approach for Vigenère systems described above could be tried, since we would have at least two rows to work with.

One-time pads seem practical in situations where one agent is communicating with a central command. They become less attractive if several agents may need to communicate with each other. The one-time feature is lost if X and Y inadvertently use the same page to talk as W and Z are using. Also capture of X's equipment makes it possible to overhear a conversation between Y and Z.

Example 3: A Transposition System

In this system, we will assume every line of the message is 63 characters long. The key is a permutation of the numbers from 1 to 63, and each line of the plaintext is rearranged using the permutation to produce the corresponding ciphertext. For example if the key is

$$1\ 11\ 21 \ldots 61\ 8\ 18 \ldots 54$$

(we would really want to use a more complicated permutation) and we use the same plaintext as in the previous two examples, we obtain:

```
TTRNRT UHOMO SFECE HYSGEH REDEN E NHS E A LE I I   CTCE   O SI
FN  ET AHBCT  DNDO AOBTRA TALOO TY IW CBEO K  SEV  H AS  TOE HE
C HNO  OWOA     S  UMOGR  TIWC  RNK   BOU S  STIT  O NF  EDTN
```

We are using the version of the plaintext including blanks. The second line of the plaintext has 55 characters, so we add 8 blanks on the end.

One method of decoding looks at a column of the ciphertext and asks what other column could immediately follow it. For example, it is possible that the column following `OBO` (the tenth ciphertext column) is `UAO` (the 8th), but the column `TFC` would yield the improbable two-letter combination `BF`.

As always, a longer message is easier to decode. Unlike simple substitution, it seems that blanks make the decoding process more difficult.

What about a known-plaintext attack? Since there is only one Y in the first line of the plaintext, we can tell that column 12 of the plaintext is

column 21 of the ciphertext, but there are other things we can't tell. In this example, there are 8 columns of three blanks at the end of the plaintext, and we can't be sure which of these corresponds to which of the all-blank ciphertext columns. (it doesn't matter for this message, but we would like to know the entire key to deal with longer plaintexts in the future) A carefully chosen plaintext can give us the entire key at once.

2 Introduction to Number Theory

2.1 Congruences

The congruence $a \equiv b \pmod{n}$ ("a is congruent to b mod n") says that, when divided by n, a and b have the same remainder.

$$100 \equiv 34 \pmod{11} \qquad -6 \equiv 2 \pmod{8}$$

In the second congruence, we are using $-6 = 8(-1) + 2$. We always have $a \equiv b \pmod{n}$ for some $0 \leq b \leq n-1$, and we are usually concerned with that b. If $a \equiv b \pmod{n}$ and $c \equiv d$, we can add or multiply

$$a + c \equiv b + d \pmod{n} \qquad ac \equiv bd \pmod{n}$$

Division does not always work: $6 \equiv 18 \pmod{12}$, but $3 \not\equiv 9$.

2.2 The Greatest Common Divisor

For a and b, the number (a, b) is the largest number which divides a and b evenly.

$$(56, 98) = 14 \qquad (76, 190) = 38$$

Theorem 1 *For any a, b there are integers x, y with $ax + by = (a, b)$*

Proof: The equation can be solved by making a sequence of simplifying substitutions:

$$\begin{aligned} 30x + 69y &= 3 \\ 30x' + 9y &= 3 \quad [x' = x + 2y] \\ 3x' + 9y' &= 3 \quad [y' = y + 3x'] \\ 3x'' + 0y' &= 3 \quad [x'' = x' + 3y'] \end{aligned}$$

It is easy to see that $x'' = 1$, $y' = 0$ is a solution to the final equation and we get a solution to the original equation by working backwards:

$$x' = x'' - 3y' = 1 \quad y = y' - 3x' = -3 \quad x = x' - 2y = 7 \quad \blacksquare$$

We could also solve an equation like $30x + 69y = 15$ by multiplying our solution: $y = -15 = 5(-3)$, $x = 35 = 5(7)$. It should be clear that the equation will have no solution in integers if 15 is replaced by something that is not a multiple of $(30, 69) = 3$.

All other integer solutions of $30x+69y = 15$ may be obtained by changing the first solution:

$$y = -15 + \frac{30}{3}t \quad x = 35 - \frac{69}{3}t \quad \text{for } t \text{ integer}$$

If we do the process illustrated on the previous page for any equation $ax+by = (a, b)$, we eventually get one of the coefficients as zero and the other as (a, b). [In fact, this process is usually presented as "Euclid's algorithm for finding the greatest common divisor."]

It is important that this process is feasible [on a computer] even if a and b are several hundred digits long. It is easy to show that the larger of the two coefficients decreases by at least $1/2$ every two equations, hence that in twenty equations the larger coefficient has decreased by $2^{-10} < 10^{-3}$, so a 600-digit number would not require more than 4000 equations. [this analysis can be improved]

We pointed out earlier that division does not work with congruences. An important application of Theorem 1 is that it does work for prime numbers.

Corollary 2 *If p is a prime number, $ar \equiv as \pmod p$, and $a \not\equiv 0$, then $r \equiv s$.*

Proof: Since p is a prime, $(a, p) = 1$, so Theorem 1 says there are integer x, y with $ax + py = 1$. Hence

$$ax \equiv 1 \pmod p \quad \text{and} \quad r \equiv (1)r \equiv axr \equiv xar \equiv xas \equiv s \pmod p \quad \blacksquare$$

Corollary 3 *If p is a prime number and $a \not\equiv 0 \bmod p$, then for any b, there is y with $ay \equiv b \pmod p$.*

Proof: We showed in the preceding proof that there is x with $ax \equiv 1 \pmod{p}$. Let $y = bx$. ∎

Corollary 4 (The "Chinese Remainder Theorem") *If $(p, q) = 1$, then for any a, b, there is an n with*

$$n \equiv a \ (mod\ p) \quad and \quad n \equiv b \ (mod\ q)$$

Proof: Theorem 1 implies there are integers x, y such that

$$px + qy = b - a \quad \text{so let } n = px + a \quad ∎$$

2.3 Powers modulo a prime

The sequence

$$a \quad a^2 \quad a^3 \ldots \quad \bmod p$$

has many applications in cryptography. Before looking at theoretical properties, the example below (done using a pocket calculator) should make clear that it is practical to compute these numbers, even when many digits are involved.

Suppose we want to compute 432^{678} mod 987. The basic trick is to start with a number and keep squaring:

$432^2 = 186624 \equiv 81 \quad 432^4 \equiv 81^2 \equiv 639 \quad 432^8 \equiv 639^2 \equiv 690 \ldots 432^{512} \equiv 858$

Since $678 = 512 + 128 + 32 + 4 + 2$,

$$432^{678} \equiv (81)(639) \ldots (858) \equiv 204 \qquad \text{(I hope!)}$$

Calculations with exponents involve not-too-many multiplications. If the numbers have several hundred digits, however, it is necessary to design special subroutines to do the multiplications (see Knuth, volume 2).

Let us look at the sequence of powers of 2 mod 11:

$$2\ 4\ 8\ 5\ 10\ 9\ 7\ 3\ 6\ 1$$

Each number from 1 to 10 appears in the sequence.

Theorem 5 *Let p be a prime. There is an a such that for every $1 \leq b \leq p - 1$, there is $1 \leq x \leq p - 1$ such that $a^x \equiv b \ (mod\ p)$.*

It is not always the case that $a = 2$. The powers of 2 mod 7 are 2, 4, 1 after which the sequence repeats and we never get 3, 5, or 6.

The proof of Theorem 5 is long, and we will give it in section 2.5. For now, we want to look at some of its consequences.

Corollary 6 *Let a be as in Theorem 5. Then $a^{p-1} \equiv 1 \pmod{p}$.*

Proof: We know that $a^d \equiv 1$ for some $1 \leq d \leq p-1$. If $d < p-1$, the sequence of powers of a would start repeating before we got all the numbers: $a^{d+1} \equiv a$, $a^{d+2} \equiv a^2$, etc. ∎

Corollary 7 *For any $b \not\equiv 0$, $b^{p-1} \equiv 1 \pmod{p}$.*

Proof: Let a be as in Theorem 5. Using Corollary 6

$$b^{p-1} \equiv a^{x(p-1)} \equiv \left(a^{p-1}\right)^x \equiv 1 \quad \blacksquare$$

Corollary 8 *If $x \equiv y \pmod{(p-1)}$, then $b^x \equiv b^y \pmod{p}$*

Proof: For some integer r, $y = r(p-1) + x$ and by Corollary 7

$$b^y \equiv \left(b^{p-1}\right)^r b^x \equiv b^x \pmod{p} \quad \blacksquare$$

Lemma 9 *Let $b \not\equiv 0$, d the smallest positive number such that $b^d \equiv 1$. Then for any $e > 0$ with $b^e \equiv 1$ d divides e evenly. In particular, by Corollary 7, d divides $p-1$ evenly.*

Proof: If d does not divide e, then $e = dr + s$ for some $0 < s < d$, but

$$b^s \equiv b^e \left(b^d\right)^{-r} \equiv 1$$

would contradict the definition of d. ∎

2.4 Primitive roots

Recall that theorem 5 says that if p is a prime, there is an a such that the equation

$$a^x \equiv b \pmod{p}$$

has a solution for any $b \not\equiv 0$. Such an a is called a *primitive root* of p, and x is called the *discrete logarithm* of b.

We showed in the beginning of section 2 that it is easy to obtain b given a and x. Finding x given a and b is much harder. Many modern encryption systems are based on the fact that no efficient way of computing discrete logarithms is known.

Lemma 10 *Let b, d be as in Lemma 9. If $d = p - 1$, then b is a primitive root.*

Proof: If $1 \leq K < L \leq p - 1$ and $b^K \equiv b^L$, then $b^{L-K} \equiv 1$. Since this can't happen by assumption, the first $p-1$ powers of b must all be different, hence must include all numbers between 1 and $p - 1$. ∎

It is often possible to find a primitive root if $p - 1$ has a small number of prime divisors. We will use $p = 1223$ as an example. $p - 1 = 2 \cdot 13 \cdot 47$. By Lemmas 10 and 9, if a is not a primitive root, then we will either have a^{26}, a^{94}, or $a^{611} \equiv 1 \pmod{1223}$. $a = 2$ and 3 fail, but $a = 5$ satisfies all three conditions, so it is a primitive root. (we could tell that $a = 4$ would not be a primitive root without testing. Why?)

It is easy to show that, if a is a primitive root, a^x is a primitive root if and only if $(x, p - 1) = 1$. In this example, this means the number of primitive roots is

$$1222 \left(\frac{1}{2}\right) \left(\frac{12}{13}\right) \left(\frac{46}{47}\right) = 552$$

Thus, if we had just chosen a at random, the probability that it would be a primitive root is $\approx .45$. Choosing a at random and testing until we found a primitive root would not be expected to take too long.

This is an example of a *probabilistic algorithm*. It is possible for it to take a long time, but the amount of time needed on average is reasonably small. We will see many other probabilistic algorithms later.

2.5 Proof of Theorem 5

This proof unfortunately requires many lemmas. It is probably a good idea to skip over the lemmas as much as possible on a first reading.

We begin with three preliminary results about greatest common divisors, which may be intuitive.

Lemma 11 *If $(a, b) = 1$ and $(a, c) = 1$, then $(a, bc) = 1$.*

Proof: By Theorem 1, there are integers x, y, w, z with $ax+by = aw+cz = 1$. Together these imply

$$ax + b(awy + czy) = 1 = a(x + bwy) + bc(zy)$$

which is impossible if $(a, bc) > 1$. ∎

Lemma 12 *If a divides b and $(b, c) = 1$, then $(ab, c) = 1$.*

Proof: Let $bx + cy = 1$ and $b = az$. Then

$$1 = bx + cy = b^2x^2 + bxcy + cy = ab(x^2z) + c(bxy + y) \quad ∎$$

Lemma 13 *If $(a, b) = 1$, cb is divisible by a, and ca is divisible by b, then c is divisible by ab.*

Proof: Let $ax + by = 1$, $cb = aw$, and $ca = bz$. Then

$$c = c(ax + by)^2 = ca^2x^2 + cb^2y^2 + 2abcxy = ab(x^2z + y^2w + 2cxy) \quad ∎$$

The key step of the proof begins with any $x \not\equiv 0$. Let d be the smallest positive number for which $x^d \equiv 1$ (there must be such a $d \leq p - 1$ since $x^K \equiv x^L$ implies $x^{K-L} \equiv 1$). By Lemma 10, if $d = p - 1$, x is a primitive root. If $d < p - 1$, we will find t with $t^i \not\equiv 1$ for all $0 < i \leq d$. If t is not a primitive root, we repeat the process, this time with $x = t$. Since the exponent d increases each time, we eventually obtain a primitive root.

Lemma 14 *There are at most d solutions to a congruence involving a polynomial of degree d:*

$$x^d + \alpha_1 x^{d-1} + \ldots \alpha_d \equiv 0 \ (mod\ p)$$

In particular, there are at most d x with $x^d \equiv 1$.

Proof: This can be proved in the same way as the corresponding theorem in ordinary algebra: if $x = \beta$ is a solution, the polynomial can be written as $(x - \beta)$ times a polynomial of degree $d - 1$, which by induction has $\leq d - 1$ solutions. ∎

We return to the proof of Theorem 5.[2] The sequence

$$x \quad x^2 \quad x^3 \ldots x^d \equiv 1 \ (\text{mod } p)$$

[2]This proof uses ideas of Michael Fischer, who pointed out an error in an earlier version.

consists of d different solutions of $x^d \equiv 1$. If $d < p-1$, let $y \not\equiv 0$ be any non-member of the sequence, with e the smallest positive number with $y^e \equiv 1$. If $e > d$, we may take $t = y$, so we will assume $e \leq d$ from now on. By Lemma 14, $y^d \not\equiv 1$, which implies e does not divide d and $e/(d,e) > 1$. We will use x and y to construct a number t such that $t^i \not\equiv 1$ for all $0 < i < de/(d,e)$. The construction is based on two lemmas:

Lemma 15 *Suppose that $(a,b) = 1$ and that a, b are the smallest positive numbers for which $u^a \equiv v^b \equiv 1$. If $k > 0$ and $(uv)^k \equiv 1$, then k is divisible by ab.*

Proof: Since $(uv)^{ka} \equiv 1 \equiv v^{ka}$, Lemma 9 implies ka is divisible by b. Similarly kb is divisible by a, and Lemma 13 gives the desired result. ∎

Lemma 16 *Let $c = (d,e)$, $d' = d/c$, $e' = e/c$. There are m, m' such that (i) $mm' = c$ and (ii) $(d'm, e'm') = 1$.*

Proof: Let m be the largest divisor of c for which $(m, e') = 1$, $m' = c/m$. The maximality of c implies $(d', e') = 1$. By Lemma 11, $(md', e') = 1$.

The maximality of m and Lemma 12 implies $(m, m') = 1$. Since $(d', e') = 1$ implies $((d', m'), e') = 1$, Lemma 11 implies $(m(d', m'), e') = 1$. Thus the maximality of m implies $(d', m') = 1$. Lemma 11 now implies $(md', m') = 1$.

Applying Lemma 11 to the conclusions of the preceding paragraphs yields $(d'm, em') = 1$. ∎

We can finally complete the proof of Theorem 5. With the notation of Lemma 16, let $t = x^{m'} y^m$. For $u = x^{m'}$, $u^a \not\equiv 1$ for $0 < a < md'$. A similar assertion holds for $v = y^m$. The other conditions of Lemma 15 are satisfied, and we can conclude that $t^k \equiv 1$ if and only if k is a multiple of $md'm'e' = de/c$. ∎

This proof does not give us an efficient procedure for finding a primitive root for large primes p, but the reason may not be obvious. The steps implicit in the lemmas *can* all be carried out efficiently. The one problem is the choice of y, which was required to be a non-member of the set of powers of x. It is too inefficient to list all the powers of x if d is large, and no significantly better way of finding a non-member is known.

3 Encryption techniques based on powers and congruences

3.1 The Diffie-Hellman key exchange procedure

A and B are communicating. C hears everything A and B say. A and B want to agree on a number, without C knowing what the number is. It may be, for example, that A and B plan to use the number as the key for future encoded messages. The procedure (also often called a *protocol*):

A and B agree on a (large) prime p and a primitive root a. These numbers are also known to C. A secretly chooses a (large) number X_1, B secretly chooses X_2. a^{X_1} and a^{X_2} mod p are publicly announced (hence known to C). The secret number will be $S = a^{X_1 X_2} \bmod p$.

$$\text{A calulates } S \equiv \left(a^{X_2}\right)^{X_1} \qquad \text{B calculates } S \equiv \left(a^{X_1}\right)^{X_2}$$

A possible drawback to this system is that neither A nor B controls what S is. If S is not a satisfactory number, they may have to repeat the protocol.

Diffie and Hellman suggest the procedure can also be used in a situation in which n people must find, for each pair of people, an agreed-upon number. For $1 \leq i, j \leq n$ the number is $a^{X_i X_j}$.

3.2 The Rivest-Shamir-Adleman public key system

A sets up a system so that anyone can send him an encoded message, but only A will be able to decode it. The message is represented as a number M. The encoding is done by a publicly known function $f(M)$, with A the only person who knows how to compute f^{-1}. A chooses two large primes p, q which he keeps secret. He announces $n = pq$ and another number d, with $(d, p-1) = (d, q-1) = 1$ (one way to do this is to choose d a prime larger than $p/2$ and $q/2$.) The encoding is

$$f(M) \equiv M^d \text{ mod n}$$

where M and $f(M)$ are both $\leq n-1$. We have seen f can be computed in a realistic amount of time even if M, d, n are many digits long.

A computes M from M^d using his knowledge of p, q. By Corollary 8,

$$\text{If } de \equiv 1 \pmod{(p-1)} \text{ then } \left(M^d\right)^e \equiv 1 \pmod{p}$$

Similarly $\left(M^d\right)^e \equiv M \pmod{q}$ if $de \equiv 1 \pmod{(q-1)}$. e satisfies these two conditions if $ed \equiv 1 \pmod{(p-1)(q-1)}$. Theorem 1 says we can let $e = x$, where x is a solution of

$$dx + (p-1)(q-1)y = 1$$

Since $\left(M^d\right)^e - M$ is divisible by p and by q, it is divisble by pq, hence we can recover M from M^d by taking to the e-th power mod pq.

It is crucial to the security of this system that knowledge of n does not allow an eavesdropper to calculate p and q. The crude approach of dividing n by all numbers up to $\sqrt{n}$ would take $\sim 10^{50}$ steps for a 100-digit n. However, many famous mathematicians have been unable to devise significantly better factoring algorithms, and this problem has been studied for at least 100 years.

One practical difficulty in using this system is the need to do calculations with many-digit numbers, especially to find primes. Another difficulty is that the inventors of this system have patented it. Amateur programmers who have posted implementations on electronic bulletin boards have received letters from "RSA Security, Inc" warning of possible patent infringement.

3.3 A public key system as hard as factoring

It is possible in theory that there is some way of computing f^{-1} for the system in the previous section that does not involve determining p and q. In the original RSA paper, the authors say

> It may be possible to prove that any general method of breaking our scheme yields an efficient factoring algorithm. This would establish that any way of breaking our scheme must be as difficult as factoring. We have not been able to prove this conjecture, however.

To see the difficulties involved in trying to prove such a thing, suppose that one could show that knowledge of a ciphertext $f(M)$ and a plaintext M enabled one to find p and q. Then one could factor n as follows:

1. Choose any M.
2. Compute $f(M)$. [Remember, we are assuming f is publicly available. Furthermore, $f(M)$ can't be too hard to compute, or the code would be impractical.]

3. Use the assumed method to obtain p, q.

In words, we are unable to distinguish between the situation in which $f(M)$ is obtained from M (easy) and the (presumably difficult) situation in which M is obtained from $f(M)$.

Rabin has suggested an alternative to the RSA system in which there is a direct connection to factoring. As in RSA, $n = pq$ is announced publicly, with primes p, q kept secret. For technical reasons, we assume $p, q \equiv 3 \pmod 4$. The encoding function is

$$f(M) \equiv M^2 \pmod n$$

The way we avoid the difficulty described above is that there are *four* numbers M_1, M_2, M_3, M_4 with $f(M_i) \equiv f(M)$. The key facts are:

1. If p, q are known, it is easy to compute all the M_i given $f(M)$.

2. If we are given n, $f(M)$, and all the M_i, we can calculate p, q.

We are *not* able to obtain p, q from just one of the M_i, so the approach based on M and $f(M)$ described above won't work. One drawback of this system is that, even with knowledge of p and q, one can only say the number sent is one of the four M_i, without being able to identify which one. In practice, this is not serious, since it is very unlikely that more than one of the M_i would correspond to a feasible message.

proof of 1: Since $p \equiv 3 \pmod 4$, there is an integer k with $4k = p + 1$. If $x \equiv (f(M))^k \pmod p$, then using Corollary 8:

$$x^2 \equiv \left(\left(M^2\right)^k\right)^2 \equiv M^{4k} \equiv M^2 \pmod p$$

Similarly if $y \equiv (f(M))^L \pmod q$ $[4L = q + 1]$, then $y^2 \equiv M^2 \pmod q$. The M_i are given from Corollary 4 by

$$\begin{array}{llll} M_1 \equiv x \pmod p & M_2 \equiv x \pmod p & M_3 \equiv -x \pmod p & M_4 \equiv -x \pmod p \\ M_1 \equiv y \pmod q & M_2 \equiv -y \pmod q & M_3 \equiv y \pmod q & M_4 \equiv -y \pmod q \end{array}$$

proof of 2: If we know the M_i, there will be two like M_1 and M_3 above with $M_1 + M_3 \equiv 0 \pmod p$ and $M_1 + M_3 \not\equiv 0 \bmod q$. Thus we can calculate $(M_1 + M_3, n)$ to obtain p.

One problem with this system is that a person with access to a "black box" that computes f^{-1} could quickly discover p, q, even if we assume that the person trying to break the code gets only one of the M_i, chosen randomly. The attacker keeps generating pairs M, $f(M)$ until he gets an M_i with $(M + M_i, n) = p$ or q.

3.3.1 Computing square roots modulo a prime

In the previous section, we assumed the primes were of the form $4n + 3$ in order to make the square root calculation as easy as possible. However, square roots can be efficiently calculated for any prime.

Let $p - 1 = 2^k e$, where e is odd. Define the sets

$$\begin{array}{ll} S_0 = \{a | a^e \equiv 1 \pmod p\} & \\ S_1 = \{a | a^{2e} \equiv 1 \pmod p\} & T_0 = S_1 \setminus S_0 = \{a | a^e \equiv -1 \pmod p\} \\ \cdots & \cdots \\ S_{k-1} = \{a | a^{2^{k-1}e} \equiv 1 \pmod p\} & T_{k-2} = S_{k-1} \setminus S_{k-2} = \{a | a^{2^{k-2}e} \equiv -1 \pmod p\} \end{array}$$

If a is a square, it must be in S_{k-1}. Thus the sets S_0 and T_i, $0 \le i \le k-2$ partition the squares. Furthermore, by starting with a^e and squaring, it is possible to identify which member of the partition includes a.

If $a \in S_0$,

$$\left(a^{(e+1)/2}\right)^2 \equiv a^e a \equiv a,$$

so it is easy to calculate a square root of a. (when $p = 4n + 3$, all $a \in S_0$)

Let b be a non-square mod p. Since $b^{2^{(k-1)}e} \equiv -1 \pmod p$ for all non-squares, it is easy to find one.

We argue by induction on i that square roots can be efficiently calculated for $a \in S_i$. We have seen that this is the case for S_0. If $a \in S_{i+1} \setminus S_i = T_i$, there is an even m with $ab^m \in S_{i-1}$. Specifically, if $m = 2^{k-1-i}$,

$$(ab^m)^{2^i e} \equiv a^{2^i e} b^{2^{k-1} e} \equiv (-1)(-1) \equiv 1$$

Since m is even, division of a square root of ab^m by $b^{m/2}$ gives a square root of a.