Lecture XIII
Cryptography

§1. Introduction

Cryptography is an crucial element in discussing security of Operating Systems. However, the development of cryptography began quite independently of any OS motivation, and is in fact an ancient subject. Our main goals here is to introduce the elements of public-key cryptography. One of the most surprising modern discoveries are the cryptographic applications of Number Theory. Since 1980, the fascinating subject of cryptography is being developed in remarkable new directions completely unpredicted by its previous history. Some background in elementary number theory will be given here.

§2. Modern Cryptography

We make a brief excursion into *cryptology* and *cryptography*, the ancient art and modern science of sending "secure" messages. As we shall see, "secure" does not necessarily mean "secret". Traditional cryptography arises in martial applications where there is an obvious need for secure communication. For instance, a general will need to communicate the latest battle plans to his commanders in the field. There is today a growing commercial need for secure data transmission, especially in electronic mail and fund transfers. In traditional encryption systems, two parties (Alice and Bob) must first agree on a common secret key K. Thereafter, if Alice wants to send Bob a secret message M, she first encrypts the message as $C = E(M, K)$ where E is the encryption function. Bob, on receipt of the encoded message C can decrypt it as $D(C, K) = M$. The pair M, C are also called the *plaintext* and *ciphertext*, respectively. Informally, we also call M, C the message and the encrypted message. Let us denote the space of *texts* (this includes plaintext or ciphertext) and space of *keys* by

$$\mathbb{T}, \qquad \mathbb{K},$$

respectively. For instance, $\mathbb{T}$ can be the set of all bit-strings, or perhaps the set of all bit-strings of length 512; $\mathbb{K}$ can be the set of bit-strings of length 256. Then the encryption E and decryption D functions are

$$E : \mathbb{T} \times \mathbb{K} \to \mathbb{T}$$
$$D : \mathbb{T} \times \mathbb{K} \to \mathbb{T}.$$

Caesar cipher. This is a very simple system attributed to Julius Caesar and well-known to children. It will serve to illustrate some of our concepts. Assume $\mathbb{T} = \Sigma^*$ where Σ is the set of integers modulo m. A key K is any integer modulo m and the encryption function simply transforms each integer i to $i+K(\mod m)$. Say $m = 37$ and messages are strings over the set of symbols

$$a, b, c, \ldots, z, \sqcup, 0, 1, \ldots, 9$$

Using this as the collating sequence (*i.e.*, $a = 1, b = 2, \ldots, z = 26, \sqcup = 27, 0 = 28, \ldots, 9 = 37$) the message

`meet␣me␣at␣7␣tonite`

can be encrypted as

`nffu0nf0bu080upojuf`

with $K = 1$.

Cryptographic Assumptions

(A) It is an axiom that plaintexts and ciphertexts are as readily distinguishable as the difference between English and gibberish. I will call this the **Fundamental Assumption** of cryptography because, without this, much of this subject would be meaningless (cryptography would reduce to pure coding theory or mathematical information theory). This assumption really says something about all natural languages, namely, they are highly structured and redundant systems. Formally, we may assume[1] an easily computable predicate

$$\mathbb{G} : \mathbb{T} \to \{English, Gibberish\}$$

that classifies all messages into plaintext (i.e., English for us) and the rest (gibberish).

Relative to some encryption function E, a ciphertext is just $E(M)$ for some plaintext M. Some properties of $\mathbb{G}$ may be postulated: it is certainly an axiom that most elements in $\mathbb{T}$ are classified as gibberish. So it is highly unlikely that a ciphertext $E(M)$ is a plaintext when M is a plaintext. Otherwise, it would be quite confusing since the transmission of ciphertext is often accompanied by plaintext. E.g., the plaintext accompanying $E(M)$ might say "This is sent by Alice to Bob".

In our applications below, we use the obvious but critical property of plaintexts: they can be "understood" by a human agent, and could have the cryptographically significant properties such as "self-authentication". What does this mean? Well, a message that tells you that a certain number divides another number is self-authenticating because you can verify if it is correct.

(B) Another basic assumption is the **insecurity of channels**, that messages can be freely intercepted[2] by a third party. Two physical models of this are the broadcast of ciphertext via radio waves or along insecure wires that can be tapped. This is the *minimal assumption* about the enemy against whom we want to secure our messages. It amounts to saying that the enemy can get hold

[1] In reality, we should allow plaintext and ciphertext to be in intermediate states arising from local errors, and which can be corrected by some other mechanism. This is not too hard in natural languages because of builtin redundancies. Error-correcting codes can be used to correct ciphertext.

[2] This third party is actually a crucial element which helps to distinguish cryptography from coding theory. In coding theory, the channel is "unreliable" but there is no malevolence. In fact, one actually relies on certain statistical properties of the so-called "unreliable" channel!

of some number of ciphertexts. Classical cryptography often stop at this level. We can postulate more powerful enemies who can launch active attacks against our system. The next level is where the enemy is in possession of some number of plaintext-ciphertext pairs. An even more powerful enemy can launch the **chosen plaintext attack**: this enemy can produce ciphertexts corresponding to any number of plaintext of his choice. In practice, it means the enemy that has gotten hold of our encryption device $E(\cdot, K)$. This is the espionage scenario during the World War II, when the Allies captured the Enigma machine used by the German military to encrypt messages.

(C) Modern cryptography assumes that the encryption and decryption schemes (E and D above) are public knowledge. Hence, modern systems must be constructed to withstand chosen plaintext attacks. In the children example, it amounts to telling the other kids in the block that we will be encoding message using Caesar cipher. In contrast, ancient cryptology often amounts to the art of dissimulation (or deception). Stories have been told of a message branded onto the shaved head of a slave who is sent to the recipient after the hair has regrown. One problem with such methods is their limited channel bandwidth, and their being not amenable to modern digital technology.

The *basic cryptographic problem* then is to discover the key K, given E, D, some ciphertexts and possibly other information. A *brute force attack* against such a system is as follows: given a few ciphertexts $C_1, \ldots, C_k$, we can try for each key $K \in \mathbb{K}$ to see if $D(C_i, K)$ is a plaintext (this is easy by assumption (A)). If K makes each C_i "plain", then we may assume the system is compromised. Hence, at the minimum, the security of a modern cryptosystem requires the size of the key space $\mathbb{K}$ to be large enough to exclude the brute force attack. For instance, $|\mathbb{K}| = 2^{512}$ may be big enough. Our Caesar cipher is insecure since it is clearly susceptible to the brute force attack.

(D) Computational complexity is the basis for defining security of modern cryptosystems. Thus, when we say above that the cryptographic attack on Caesar cipher is "easy" we meant that the simple brute force attack is computationally easy. Note that the Fundamental Assumption of cryptography is necessary even here. We may identify "easy" with polynomial time, perhaps in the randomized sense. In particular, we assume that the encryption and decryption functions E, D must be polynomial time computable.

Other properties will be invoked as we go along. The bottom line is that cryptography has not yet reached a maturity that permits a systematic logical analysis of all its concepts.

Exercises

Exercise 2.1: What properties of English justifies the Fundamental Assumption of cryptography? ◊

Exercise 2.2: Suppose that we extend Caesar cipher so that we operate on a block of $m \geq 1$ symbols at a time. Treating each block as a new symbol, we have effectively increased the size of the key space from 37 to 37^m. How can we implement a form of Caesar cipher on this enlarged alphabet? How large can you reasonably choose m? Discuss the security of this "block Caesar cipher"? $\diamondsuit$

End Exercises

§3. Public-key cryptography

An unfortunate property of the traditional crytographic systems is the need for prior secure communication between Alice and Bob (to agree on the secret K). In the mid-1970's, Whitfield Diffie and Martin Hellman [**?**] proposed the concept of a *public-key cryptosystem*. See [**?**] for a non-technical account of this development, including the role of Ralph Merkle. In such a system, a participant Alice chooses a pair of keys S_A, P_A where S_A is the *secret key* known only to Alice and P_A is the *public key* known to all. We can postulate a *public key directory* in which the public keys of all participants are published. Corresponding to S_A, P_A are a pair of permutations,

$$\widehat{S}_A, \widehat{P}_A : \mathbb{T} \to \mathbb{T}$$

with the properties

1. $\widehat{S}_A(\widehat{P}_A(M)) = \widehat{P}_A(\widehat{S}_A(M)) = M$ for all $M \in \mathbb{T}$.
2. If $A \neq B$ then $\widehat{S}_A(\widehat{P}_B(M))$ and $\widehat{P}_A(\widehat{S}_B(M))$ is gibberish for most M. *Á fortiori*, they are both unequal to M for most plaintexts M.
3. $\widehat{S}_A$ (resp., $\widehat{P}_A$) is easy to compute if we know S_A (resp., P_A), and difficult otherwise. Such functions are called *trapdoor functions*.

Depending on the application, there are various cryptographic requirements for sending and receiving messages. These requirements amounts to having security in the face of various forms of *cryptographic attacks*. Some of these requirements can be satisfied within the public-key cryptosystem, as we now indicate. Besides the protagonists Alice and Bob, we further postulate (a) some third party such as Carol or Carl who are generally unfriendly to Alice or Bob, and (b) a judge Dee or judge Dick who can arbitrate between Alice and Bob. Note that traditional cryptography does not have to deal with (b).

Protocol A: receiver authentication. Let us see how Alice can send Bob a message M in such a system. Alice looks up the public key directory and finds P_B. She then sends Bob the encrypted message

$$\widehat{P}_B(M).$$

Bob on receipt of $\widehat{P}_B(M)$ can recover the message by applying $\widehat{S}_B$ to $\widehat{P}_B(M)$. Call this *protocol A*. In protocol A, Alice is assured that only Bob is able to read the message (no one else can compute $\widehat{S}_B$ with ease). In other words, protocol A "authenticates the receiver" since the plaintext M remains secret to other third parties.

Protocol A may be useful if Alice wants to send the FBI (=Bob) some secret. Alice does not have to reveal her identity to the FBI with this protocol, and in fact, it is impossible for Alice to convince anyone that she was in fact the sender. Presumably in this application, it is not important for the FBI to know who sent the message. The FBI does not need to trust the message, but it is important that message is **self-authenticating**. Such a message may say: `The murder weapon is inside a PC at 251 Mercer Street`.

Using an OS example, if Bob forgot his password and asks the password administrator for his password, the system administrator can use Protocol A.

Protocol B: sender authentication. Suppose Alice wants to send a message that *authenticates her as the sender*. Protocol A cannot do this. It is no use to claim in the plaintext that "`I, Alice, sent you, Bob, this`" since Carol or Carl could well have posed as Alice in this way. A solution is for Alice to send the encrypted message

$$\widehat{S}_A(M),$$

which Bob can decrypt by applying $\widehat{P}_A$ to the received message. If the result is a plaintext, then Bob has "authenticated Alice as the sender". Call this *protocol B*. Note that this encrypted message ought to be accompanied by a plaintext that says: "Alice sends you this".

Sender authentication solves the *imposter problem* or *masquarading problem*. Protocol B might be useful to the military commander who wants to send a message to his men to lay down their arms. The message is public but it is important that each receiver can authenticate the commander as sender.

An OS application is where a computer is asked by a remote user to execute some secure command, such as to shut down. If the computer can authenticate the sender of the command, the secure command may be executed.

Digital signature. As in human handwritten signatures, "digital signatures" are meant to authenticate the sender. However, digital signature must be message-specific in the sense that it cannot be transferred to another message. Otherwise someone in possession of a message signed by Alice can reuse it to forge a different message with Alice's signature. Note that traditional signatures is message-specific since it is written on the paper containing the message, and assuming it is not easy to transfer the signature to a different piece of paper. In the electronic medium, the latter is trickier to achieve. Protocol B implements a form of digital signature in the case of sending a public message.

Reneging. Protocol B can be modified to solve the problem of *reneging*: namely Bob can subsequently prove to a judge that (1) he received a message M intended for Bob and (2) M came from Alice. In other words, we want both sender and receiver authentication. Protocol B already achieves property (2).

To achieve property (1), we modify protocol B so that Alice says explicitly in the message M that it is addressed to Bob. [Bob will not act upon any message that does not have this in plaintext.] This trick is interesting: we have resorted to a non-mathematical property of plaintexts to achieve cryptographic security.

Private Bob on receiving the message "`I, commander Alice, asks all soldiers to surrender to the enemy at once`" under this modified protocol B can safely act on this message. Of course, this only works if the message can be public.

Protocols AB and BA. Suppose the message that Alice sent to Bob is meant for him alone, and Bob must know that the message came from Alice. Unlike the previous solution to the reneging problem, we now want the message to remain secret to Alice and Bob. This situation might arise if Alice is the client of Bob the stockbroker. Bob does not want to be liable for acting on Alice's instructions if Alice can later deny having sent him the message. [If Alice told Bob to sell all her IBM shares immediately, and the next day, IBM shares increased in value, there is incentive for Alice to deny having given the instructions.]

Here are two other protocols to solve this reneging problem for secret messages. Alice could send

$$\widehat{P}_B(\widehat{S}_A(M)) \quad \text{or} \quad \widehat{S}_A(\widehat{P}_B(M)).$$

Bob can decode both messages, in the obvious way. Call these *protocol AB* and *protocol BA*, respectively. Is there a significant cryptographic difference between the two protocols? There are many subtle differences, although both can be used to achieve the same purposes with some care.

In protocol AB, only Bob could decode the sent message to $\widehat{S}_A(M)$. This authenticates the receiver as Bob. Is the sender really authenticated? Of course, no one but Alice could produce a string of the form $\widehat{S}_A(M)$. But Carl could have stolen $\widehat{S}_A(M)$ (say by wiretapping Alice's line) and forwarded it to Bob. Recall that we assume that wiretaping is freely available to Carl. So the only way that Bob can authenticate Alice as sender and know that he was the intended recipient is (similar to our original reneging solution) for Alice to "sign her name and explicitly address Bob" in plaintext M.

What about protocol BA? It seems that Bob can be sure that Alice sent the message and surely it was addressed to him. But it is open to another cryptographic attack.

Forgery or Fabrication. The reneging problem above asks for a protocol for sending messages so that senders cannot later deny having ever sent them. The converse is the *forgery problem*: can Carl forge a message to Bob (by posing as Alice or by modifying an actual message from Alice to Bob)? [The former is the imposter problem.] A protocol is *immune to forgery* if the message (ciphertext) sent cannot be used or modified for use in a forgery.

Consider protocol BA again. The imposter Carl could intercept the ciphertext $\widehat{S}_A(\widehat{P}_B(M))$, decode it into $\widehat{P}_B(M)$. Although Carl could not read the message M, he could resend the message $\widehat{S}_C(\widehat{P}_B(M))$ to Bob. Of course, this may be silly for Carl to do since he does not know the contents of M, nor could he modify M. But ensure that Carl does not attempt this, Alice should again sign her name and explicitly address Bob.

Judge Dee and Judge Dick. We refine the receiver authentication problem. *Simple receiver authentication* means that Bob can convince himself that the message was meant for him. *Strong receiver authentication* means he can convince a judge as well. When is the strong version really different than simple version? Let us assume protocol A is used. The difference between Bob convincing himself, versus Bob convincing a judge, is that Bob knows S_B but the judge does not. To convince the judge that the message was intended for him, Bob has to reveal the secret key S_B to the judge. If for some reason, Bob does not wish to reveal S_B to the judge, this protocol does not solve the strong version. For this reason, we may distinguish between two kinds of judges: judge Dee is trusted by all parties and judge Dick is not necessarily to be trusted by any party. So protocol A is a solution under judge Dee but not judge Dick.

Similarly, we can have *simple sender authentication* and *strong sender authentication*. Again, the difference is between Bob convincing himself versus convincing a judge. Assuming the use of protocol B, to convince a judge that a message was sent by Alice, Bob must reveal to the judge the contents of the message. Again, this would work only if we have judge Dee and not Dick.

Summary of cryptographic parameters. The preceding scenarios indicate some requirements of cryptographic protocols. Messages can be public or secret (private). Private messages can weak (if we can share it with a judge) or strong kind (if we cannot). Authentication of receiver or of sender come in several flavors. The sender and receiver can be mutually trusting or mutually distrusting, or perhaps there is only one-sided mistrust. The communication takes place in an benign or hostile environment. The benign environment still poses threats if the two communicating parties are not mutually trusting – this means that cryptographic issues can still arise between Alice and Bob in the absence of Carl or Carol. There are degrees of hostility. Two weak forms of hostility are (1) there the communication channel is simply unreliable in a random manner, and (2) there is an enemy who can only passively intercept messages. But we may endow the enemy with more power, such as the ability to send deceptive messages which are perhaps based on previously intercepted messages. Note that in the above examples, a judge could well be an algorithm or a blackbox that can check certain inputs. We can have various kinds of judges: some judges are omnipotent (this amounts to having unlimited computational power), some judges are trusted by only some of the parties in a dispute. For instance, if an omnipotent judge sees a ciphertext of the form $S_A(P_B(S_A(M)))$, it should be enough to convince the judge that M was sent by A to B.

Finally, there are other directions to extend these concepts: for instance, multiple communicating parties. In many applications (say in electronic commerce), we often need to have mechanisms for deferral of trust, leading to the idea of "trust management".

Exercises

Exercise 3.1: Here is another protocol for digital signatures: if Alice wants to send Bob the plaintext M, she first computes $M' := \widehat{S}_A(M)$ and sends Bob the "signed message" (M, M') using protocol A, with M' constituting the signature. That is, she sends Bob the ciphertext $\widehat{P}_B(M, M')$. Now Bob can decrypt this back to a pair (M, M'). To verify that M was sent by Alice, he can compute $\widehat{P}_A(M')$ and see that it is equal to M. Note that Bob must somehow know that it was sent by Alice in order to know the function $\widehat{P}_A(M')$ to use. One way is for Alice to announce herself in plaintext in M. Compare this protocol with our protocol AB, and describe any crytographic differences. ♢

Exercise 3.2: Receiver authentication is a 3-ary relation $AuthenRec(A, B, M)$ indicating that A is convinced that B sent A the message M. It can be broken up into two parts: A is convinced that B sent M and A is convinced that M is addressed to A. These can further be analyzed into more basic forms: let $Know(A, f)$ denote that A knows or is convinced of fact f. The elementary facts in this case are $SentBy(M, B)$ and $SentTo(M, A)$. So:

$$AuthenRec(A, B, M) \equiv Know(A, SentBy(M, B)) \wedge Know(A, SentTo(M, A)).$$

(i) Do a similar, logical analysis for the other concepts discussed above.
(ii) Discuss some properties ("axioms") of the "*Know*" modality and the elementary facts. ♢

Exercise 3.3: We can also distinguish between two kinds of secret messages. Those that can be revealed to any judge, and those that cannot. Let us call them *weak* and *strong secrets*, respectively. How is this viewpoint different from the two kinds of judges? ♢

Exercise 3.4: Consider other forms of the reneging problem. In the previous reneging problem, Bob presumaby does not have the cooperation of Alice. Consider a scenario where *both* Alice and Bob are eagar to convince the judge that Alice is the sender of the message to Bob. That is, Alice is cooperative with Bob here. When does such a scenario arise? What cryptographic consequences follow from this in the context of our above discussions? ♢

Exercise 3.5: For each of the above scenarios, analyze and make explicit (i) the *requirements* of the problem, and (ii) the *assumptions* on the public-key cryptosystem. ♢

Exercise 3.6: Discuss the class of "alternating protocols", protocol $ABA \ldots A[B]$ or protocol $BAB \ldots B[A]$. ♢

Exercise 3.7: Investigate multiparty communication protocols. ♢

End Exercises

§4. Some Number Theory

Before presenting a specific public-key cryptosystem, we need some basic facts from elementary number theory. They can be found in most elementary texts on the subject; especially recommended are Hardy and Wright [**?**] and Hua [**?**]. "Number" in this section always means a natural number $n \in \mathbb{N}$.

Divisibility. The concept of divisibility is the starting point of number theory: let $m, n \in \mathbb{Z}$. We say m **divides** n, or n is a **divisor** of m, denoted $m|n$, if there is some $a \in \mathbb{Z}$ such that $n = ma$. Non-divisibility of n by m is indicated by $m \not| n$. Note that every integer divides 0; but O divides only itself. A **trivial** divisor of n is a divisor m such that $|m| = n$ or $|m| = 1$. The **division property** is this: given $a, b \in \mathbb{Z}$, there are unique $q, r \in \mathbb{Z}$ such that

$$a = qb + r, \qquad (0 \le r < |b|).$$

We call q and r the *quotient* and *remainder of a divided by b*. The algorithm to compute this q and r is called the **division algorithm**. We shall also indicate the remainder using the infix notation $r = (a \,\mathbf{mod}\, b)$.

Primes. A number n is *prime* if it has exactly two trivial divisors. A non-zero number is *composite* it is at least one non-trivial divisor. Thus $0, 1$ are neither prime nor composite; 2 is the smallest prime and the only even prime. Usually, we let p denote a prime. The *Fundamental Theorem of Arithmetic* says that every number $n \ge 2$ has a unique representation of the form

$$p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}, \qquad (k \ge 1)$$

where $p_1 < p_2 < \cdots < p_k$ are primes and $e_i \ge 1$ for all i.

Greatest common divisors. The *greatest common divisor* $\mathtt{GCD}(m, n)$ of two integers m, n are defined as follows:

$$\mathtt{GCD}(m,n) = \begin{cases} 0 & \text{if } \ m = n = 0 \\ \text{largest common divisor of } m, n & \text{else.} \end{cases}$$

When there is no confusion, we prefer to write (m, n) for $\mathtt{GCD}(m, n)$. Note that $(m, n) = (n, m)$ and $(m, 0) = |m|$ and $0 \le (m, n) \le \max\{|m|, |n|\}$. We say m, n are *relatively prime* (or m is *relatively prime to* n) if $(m, n) = 1$. Alternatively, we say *co-prime* instead of "relatively prime".

Euclidean algorithm. Given $m \ge n \ge 1$, the Euclidean algorithm computes the GCD of m, n via the **remainder sequence**

$$m_0, m_1, \ldots, m_k, m_{k+1} = 0, \qquad (k \ge 1)$$

where $m_0 = m, m_1 = n$ and for $i \ge 1$,

$$m_{i+1} = (m_{i-1} \,\mathbf{mod}\, m_i). \tag{1}$$

Then m_k is the GCD of m, n. We will shortly give a proof of this fact. Schönhager (1971) has shown how to implemented this algorithm in time $M(\log m) \log\log m$, where $M(b) = \mathcal{O}(b \log n \log\log b)$ is the time to multiply two b-bit integers.

The **Extended Euclidean algorithm** on m, n computes a pair of integers s, t such that

$$sm + tn = (m, n).$$

More generally, suppose we want to relate each remainder m_i to m, n using the *co-factors* s_i, t_i:

$$m_i = s_i m + t_i n. \tag{2}$$

The Euclidean algorithm can be extended to compute two auxilliary **co-factor sequences**:

$$(s_0, s_1, \ldots, s_k, s_k), \qquad (t_0, t_1, \ldots, t_k, t_k).$$

For $i = 0$ and $i = 1$, these co-factors are easy to specify:

$$s_0 = 1, \quad t_0 = 0, \qquad s_1 = 0, \quad t_1 = 1$$

For $i \ge 1$, it is not hard to see that the invariant (2) is preserved if we define

$$s_{i+1} = s_{i-1} - s_i q_i, \quad t_{i+1} = t_{i-1} - t_i q_i$$

where $q_i = \lfloor m_{i-1}/m_i \rfloor$. Indeed, we may rewrite the basic step (1) of the original Euclidean algorithm in the same form:

$$m_{i+1} = m_{i-1} - m_i q_i. \tag{3}$$

In summary: we succinctly describe the generic step of the extended algorithm as follows: for $i \ge 1$,

$$\boxed{\begin{array}{l} q_i \leftarrow \lfloor m_{i-1}/m_i \rfloor; \\ (m, s, t)_{i+1} \leftarrow (m, s, t)_{i-1} - q_i (m, s, t)_i. \end{array}}$$

Let us now prove the correctness of the extended Euclidean (and hence the plain Euclidean) algorithm. Namely, $m_k = (m, n)$. From the relation $m_k = s_k m + t_k n$, we conclude that

$$(m, n) | m_k. \tag{4}$$

Conversely, $m_{k-1} = q_k m_k + m_{k+1} = q_k m_k$ implies $m_k | m_{k-1}$. From equation (3), we see that if m_k divides two consecutive members m_i, m_{i+1} of the remainder sequence, then it divides the preceding member m_{i-1}. Since m_k divides m_{k-1} and m_k, we conclude it divided m_{k-2}. Repeating this argument, we finally see that m_k divides every member of the remainder sequence. In particular, m_k divides $m_1 = n$ and $m_0 = m$. Hence $m_k|(m,n)$. This, together witn (4) shows that $m_k = (m,n)$, as desired.

The extended Euclidean algorithm shows that the GCD of m, n can be written as a linear combination of m and n: $sm+tn = (m,n)$. Note that if $(m,n) = 1$ then $sm+tn = (m,n) = 1$ means that $sm \equiv 1(\mod n)$, *i.e.*, s is the inverse of m modulo n. We conclude: *if m, n are co-prime then m has an inverse modulo n and this can be computed by the extended Euclidean algorithm.*

Residue systems. Let n be a number and a, b be integers.

- We say a is **congruent to b modulo n**, and writes (following Gauss),

  $$a \equiv b(\mod n),$$

  if $m|(a-b)$. Alternatively, $a \equiv b(\mod n) \iff (a \,\mathbf{mod}\, n) = (b \,\mathbf{mod}\, n)$. For fixed n, $\equiv$ is an equivalence relation. A *complete residue system modulo n* is a set of integers that contains exactly one representative from each congruence (or equivalence) class. The *additive group modulo n*,

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

  is a complete residue system modulo n. Note that $\mathbb{Z}_0 = \emptyset$ (the empty set). Another complete residue system modulo n is $\{-\frac{n-1}{2}, -\frac{n-3}{2}, \ldots, -1, 0, 1, \ldots, \frac{n-1}{2}\}$ for odd n. For $a \in \mathbb{Z}_n$, we can write $-a$ for $n-a$ modulo n; clearly $(-1)(-a) \equiv a(\mod n)$ and $(-1)a \equiv (-a)(\mod n)$. We also use the notation

  $$\mathbb{Z}_n^+ := \mathbb{Z}_n - \{0\}.$$

- The power of congruence is that we can often treat them like equalities. Some useful properties of congruence include:

  $$a \equiv b(\mod n). \Rightarrow .\frac{a}{d} \equiv \frac{b}{d}(\mod \frac{n}{d})$$

  where $d = (a, b, b)$. Again,

  $$ma \equiv mb(\mod n). \Rightarrow .a \equiv b(\mod \frac{n}{d})$$

  where $d = (m, n)$. For instance,

  $$2^n \equiv 2(\mod n). \Rightarrow .2^{n-1} \equiv 1(\mod n)$$

  provided n is odd.

- If a and n are co-prime and $a \equiv b(\mod n)$ then it easily seen that b and n are also co-prime. Any property such as co-primeness that holds uniformly for equivalent numbers modulo n is called a *modulus n property.* A *reduced residue system modulo n* is a set of integers that contains exactly one representative from each equivalence class that is co-prime to n. The *multiplicative group modulo n*,

$$\mathbb{Z}_n^* := \{a \in \mathbb{Z}_n : (a, n) = 1\},$$

  is a reduced residue system modulo n. Again, $\mathbb{Z}_0^*$ is the empty set. Also, $\mathbb{Z}_1^* = \{0\}$ (based on a technicality, we might say). It is easy to see that $\mathbb{Z}_n^*$ is closed under multiplication modulo n; it is a group because the extended Euclidean algorithm above shows that every element in $\mathbb{Z}_n^*$ has an inverse. For instance, $\mathbb{Z}_6^* = \{1, 5\}$ and $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$. In general, for $n > 1$, $\mathbb{Z}_p^* = \mathbb{Z}_p^+$ iff n is prime.

- We call a a *quadratic residue* of n if there exists an x such that $a \equiv x^2(\mod n)$; otherwise a is a *quadratic non-residue*. Thus 1 is always a quadratic residue of n. We may say quadratic residues are those elements with square-roots. Again, we see that quadratic residuosity of a number modulo n is a modulus n property.

Euler totient function. The *Euler totient function* $\phi(n)$ is defined as $\phi(n) = |\mathbb{Z}_n^*|$, *i.e.*, $\phi(n)$ is the number of co-prime equivalence classes modulo n. We thus have

$$\phi(0) = 0, \quad \phi(1) = 1, \quad \phi(2) = 1, \quad \phi(3) = 2, \quad \phi(4) = 2.$$

The following says that ϕ is a "multiplicative function":

Fact 1.

$$(m, n) = 1 \quad \text{implies} \quad \phi(mn) = \phi(m)\phi(n).$$

We leave the proof as an exercise.

Fact 2. *For p prime and $e \geq 1$,*

$$\phi(p^e) = p^{e-1}(p - 1).$$

In proof, note that if $q = p^e$ then $x \in \mathbb{Z}_q - \mathbb{Z}_q^*$ iff $p|x$. But there are exactly p^{e-1} such values of x, so $|\mathbb{Z}_q^*| = |\mathbb{Z}_q| - p^{e-1} = p^{e-1}(p - 1)$.

Using the last two facts, we obtain a simple formula for $\phi(n)$ provided we have a factorization of n. For instance, if $n = 2^5 \cdot 3 \cdot 5^3$ then $\phi(n) = 2^5(2 - 1) \cdot 2 \cdot 5^2(5 - 1) = 2^8 5^2$.

Fact 3 (Fermat-Euler Theorem).

$$(m, n) = 1 \quad \text{implies} \quad m^{\phi(n)} \equiv 1(\mod n).$$

In proof, suppose $a_1, \ldots, a_{\phi(n)}$ is a reduced system modulo n and $(m,n) = 1$. Then we see that $ma_1, \ldots, ma_{\phi(n)}$ is also a reduced system modulo n. Thus, modulo n, we have

$$\prod_{i=1}^{\phi(n)} a_i \equiv \prod_{i=1}^{\phi(n)} ma_i \equiv m^{\phi(n)} \prod_{i=1}^{\phi(n)} a_i.$$

But each a_i is invertible and so can be can cancelled in this equivalence, giving $1 \equiv m^{\phi(n)} (\mod n)$, as desired.

Corollary 1 (Fermat's little theorem). *If p is prime, $m^{p-1} \equiv 1(\mod p)$ for $m \in \mathbb{Z}_p^+$.*

Solving Linear Congruences. Consider the following congruence

$$12x \equiv 9 \mod 15 \tag{5}$$

to be solved for x. It is easy to see that $x = 2$ is a solution. Next consider solve the following:

$$12x \equiv 5 \mod 15.$$

It may take a moment of searching to conclude that this congruence has no solution. [In proof, suppose it has a solution. Then $12x + 15z = 5$ for some $z \in \mathbb{Z}$. Since 3 divides the left-hand side $12x + 15z$, it must divide the right-hand side 5. This is a contradiction.] In general, we want to solve the linear congruence

$$ax \equiv b(\mod n) \tag{6}$$

where $a, b \in \mathbb{Z}_n$. The following lemma tells us when a solution exists.

Lemma 2. *Let $d = (a,n)$. The congruence (6) is solvable if and only if $d|b$.*

Proof. ($\Rightarrow$) The argument proceeds as in the example above. Suppose (6) has a solution. Then $b = ax + nz$ for some $z \in \mathbb{Z}$. Since d divides the right-hand side $ax + nz$, it must divide the left-hand side b.
($\Leftarrow$) Suppose $d|b$. Then we can express a, b, n as

$$a = a'd, \quad b = b'd, \quad n = n'd.$$

Consider the modified congruence

$$a'x' \equiv b'(\mod n').$$

Multiplying both sides by $(a')^{-1}$, the inverse of a' modulo n', we obtain $x' \equiv b'(a')^{-1}(\mod n')$. Thus $x' = b'(a')^{-1} + n'z$ for some $z \in \mathbb{Z}$. If we multiply this equation by $a = a'd$, we obtain $ax' = a'db'(a')^{-1} + a'dn'z = b + a'nz$. This shows that x' is a solution to (6). **Q.E.D.**

As a corollary of the proof, we get

Corollary 3. *If (6) has a solution, then the solution is given by*

$$x = (b'(a')^{-1} \,\mathbf{mod}\, n')$$

where $a' = a/d$, $b' = b/d$, $n' = n/d$ *and* $(a')^{-1}$ *is the inverse of* a' *modulo* n'.

In example (5), $d = (12, 15) = 3$ and hence $a' = 12/3 = 4$, $b' = 9/3 = 3$ and $n' = 15/3 = 5$. Hence $(a')^{-1} = 4$ since $a' \cdot 4 \equiv 1(\mod 5)$. The solution is therefore $x = b'(a')^{-1} \,\mathbf{mod}\, n' = 3 \cdot 4 \,\mathbf{mod}\, 5 = 2$.

Next, notice that the (5) in fact has three distinct solutions: $x = 2, 9, 14$. In general, we have:

Lemma 4. *The equation* $ax \equiv b(\mod n)$, *if it has any solutions, has exactly* $d = (a, n)$ *solutions. In fact, modulo* n, *these* d *solutions are congruent to*

$$x_0 + in/d, \quad (i = 0, 1, \ldots, d-1)$$

where x_0 *is any solution.*

Proof. From the corollary, we know that if there is a solution, then $d|b$. As before, let $b = b'd, a = a'd, n = n'd$. For any solution x_0, write

$$x_i := x_0 + in' \ (i = 1, 2, \ldots, d-1).$$

It is easy to check that each x_i is also a solution: $ax_i = ax_0 + ain' \equiv b + a'in \equiv b(\mod n)$. We note that each of the x_i's are also distinct solutions modulo n. To see this, suppose $0 \le i < j < d$ and $x_i \equiv x_j(\mod n)$. This implies $n|(x_i - x_j)$ or $n|(i-j)n'$. This gives the contradiction that $d|(i-j)$ since $i - j < d$.

Finally, we must show that there are no other solutions. Let $y = x_i + \delta$ $(0 < \delta < n')$ be another solution (modulo n, any other solution must have this form for some i). Thus $ay \equiv b + a\delta \equiv b(\mod n)$. This means $n|a\delta$, or $n'|a'\delta$. Since $(a', n') = 1$, this means $n'|\delta$, contradiction. **Q.E.D.**

Finite Groups. We have introduced two finite groups: the additive group $\mathbb{Z}_n$ and the multiplicative group $\mathbb{Z}_n^*$. Lagrange's index theorem for finite groups says that if H is a subgroup of G then $|H|$ divides $|G|$. We call $|G|/|H|$ the *index* of H in G, denoted $G : H$.

Fact 4. $\mathbb{Z}_n^*$ *is a cyclic group iff* $n = 2, 4, p^e$ *or* $2p^e$, *where* p *is an odd prime and* $e \ge 1$.

A generator $g \in \mathbb{Z}_n^*$ of the cyclic group is called a *primitive root of* n. If g is such a generator, then the *discrete logarithm* or *index* of $x \in \mathbb{Z}_n^*$ (*to base* g, *modulo* n) is the number $i \in \mathbb{Z}_n^*$ such that $g^i \equiv x(\mod n)$. We write $\mathrm{ind}_{g,n}(x) = i$ for this index. If g, n are understood or irrelevant, we simply write $\mathrm{ind}_n(x)$ or $\mathrm{ind}(x)$.

Fact 5 (Discrete logarithm theorem). *If* g *is a primitive root of* n *then* $g^x \equiv g^y(\mod n)$ *iff* $x \equiv y(\mod \phi(n))$.

Chinese remainder theorem. Let $m = \prod_{i=1}^k m_i$ where the m_i are co-prime in pairs. Define the map

$$f : \mathbb{Z}_m \to \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_k}$$

where $f(b) = (b_1, \ldots, b_k)$ where $b \equiv b_i(\mod m_i)$ for all i. The Chinese remainder theorem says that f is an isomorphism of rings:

$$\begin{aligned} f(-b) &= -f(b) \\ f(b+c) &= f(b) + f(c) \\ f(b \cdot c) &= f(b) \cdot f(c). \end{aligned}$$

The right-hand side of these equations needs to be clarified since they describe operation on k-vectors. The operations are to be applied in a componentwise manner, and for the ith component, the operations are modulo m_i. Both f and f^{-1} can be computed in polynomial time.

Exercises

Exercise 4.1:
(i) Say that $a \in \mathbb{Z}_p^*$ *belongs to exponent* m if, modulo p, $a^m \equiv 1$ and $a^i \not\equiv 1$ for $i = 1, \ldots, m-1$. Give a polynomial time algorithm to determine the exponent of any number $a \in \mathbb{Z}_p^*$. Thus we can verify in polynomial time if a is a primitive root of p.
(ii) Give a method to find a primitive root of p. HINT: if a belongs to exponent m and a' belongs to exponent m' and $m' \le m < p$, find b belonging exponent $> m$. ♢

Exercise 4.2: (a) The Fibonacci numbers, $F_n, n \ge 0$, are defined by the recurrence

$$F_{n+1} = F_n + F_{n-1}, \quad n \ge 1, \quad F_0 = 0, F_1 = 1.$$

Clearly F_n can be evaluated, by using this recurrence, in $T(n) = O(n)$ arithmetic operations. Show a method that is asymptotically faster than $O(n)$. What is $T(n)$ for your method? HINT: let $M = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. How are the entries in M^n related to the Fibonacci numbers?
(b) Given two numbers $\tilde{F}_0 > \tilde{F}_1 > 0$ we can define the sequence $\tilde{F}_2, \tilde{F}_3, \ldots$ where $\tilde{F}_{n+1} = \tilde{F}_{n-1} - \tilde{F}_n$. This defines a decreasing sequence of numbers. We want an algorithm to check if there exists an $n > 0$ such that $\tilde{F}_n = 1$ and $\tilde{F}_{n+1} = 0$. If such an n exists, the algorithm returns this n, otherwise it returns 0. Note that if such an n exists, then $\tilde{F}_i = F_{n-i}$ for all $i = 0, \ldots, n$. Describe a method that has the same asymptotic complexity as in PART (i). HINT: $N = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix}$ is the inverse of the matrix M. ♢

End Exercises

§5. The RSA cryptosystem

So far, we have been using facts about public-key systems without actually showing that they exist. We now present one such cryptosystem which was suggested in 1976 by Rivest, Shamir and Adleman [?]. It is necessary to assume that $\mathbb{T} = \mathbb{Z}_N$ for some arbitrary but large integer N. For instance, N may be 2^{300}. This is not a serious restriction, since if the user has a larger message, it can be broken down into blocks of $\lg N$ bits. The pair of keys S_A, P_A that a prospective participant (call her Alice) must choose is obtained as follows:

- Pick a pair of primes p, q such that $pq > N$ (in our example where N has 300 bits, we can let p, q each be more than 150 bits).
- Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$.
- Pick an odd number $e < \phi(n)$ such that $\mathtt{GCD}(e, \phi(n)) = 1$. This is easy: randomly pick an odd number e' less than $\phi(n)$, and compute the $d = \mathtt{GCD}(e', \phi(n))$. Then let $e = e'/d$.
- Compute d, the multiplicative inverse of $e \mod \phi(n)$.
- Now set $P_A := (e, n)$ and $S_A := (d, n)$. We define the permutations $\widehat{P}_A, \widehat{S}_A$ as follows:

$$\widehat{P}_A(M) = M^e \,\mathbf{mod}\, n, \qquad \widehat{S}_A(M) = M^d \,\mathbf{mod}\, n$$

  for all $M \in \mathbb{T}$. Note that 'e' and 'd' denote (respectively) 'encode' and 'decode'.

It is not hard to see that if n has B bits then the encoding and decoding functions can be performed in $O(B^3)$ time. Why are the encoding and decoding functions inverses of each other? To see this,

$$\begin{aligned}
\widehat{P}_A(\widehat{S}_A(M)) &= \widehat{S}_A(\widehat{P}_A(M)) \\
&= (M^{ed} \,\mathbf{mod}\, n) \\
&= (M^{1+c\phi(n)} \,\mathbf{mod}\, n) \quad \text{(for some } c \in \mathbb{Z}). \\
&\equiv M \mod n.
\end{aligned}$$

This equivalence follows from the Fermat-Euler theorem if $(M, n) = 1$. What if $(M, n) > 1$? In that case, we claim:

$$M^{1+c\phi(n)} = M^{1+c(p-1)(q-1)} \equiv M(\mod p)$$

There are two cases: if $(M, p) = 1$, this follows by Fermat's little theorem, and if $p|M$ then the asserted equivalence is the trivial $0 \equiv 0(\mod p)$. The same argument holds when we replace p with q: $(M^{ed} \,\mathbf{mod}\, n) \equiv M(\mod q)$. By the Chinese remainder theorem, we conclude that $(M^{ed} \,\mathbf{mod}\, n) \equiv M(\mod n)$. Since $0 \le M < n$, this implies $(M^{ed} \,\mathbf{mod}\, n) = M$, as desired.

Worked Example. Let $N = 2^{10}$. To use this system, the user must pick a pair p, q of primes, each at least 5 bits long (bigger than 32).

Suppose Alice picks $p = 47$ and $q = 59$, so that $n = 2773$ and $\phi(n) = 2668$. She then chooses $d = 27$. Using the extended Euclidean algorithm, he computes $e = 1087$. Thus, his public key is $(n, e) = (2668, 1087)$ and his private key is $(n, d) = (2668, 27)$.

Similarly, Bob picks $p = 37$ and $q = 31$ so that $n = 1147$ and $\phi(n) = 1080$. He then chooses $(d, e) = (7, 463)$.

Now, Alice wishes to send the message 294 to Bob. She first applies her secret key to encode 294 as $294^{1087} \mathbf{mod}\, 2773 = 1061$. Then she applies Bob's public key to encode 1061 as $1061^{463} \mathbf{mod}\, 1147 = 391$.

When Bob receives 391, he applies his secret key and then Alice's public key to recover the message. Check that Bob will recover the message sent.

Other Issues.
(A) Security of RSA. In what sense is the RSA system secure? It is clearly not absolutely secure against an enemy that has "exponential" computing power, or enough computing power to factor numbers. This is because the secret key can be determined once we factor $n = pq$. Basically, proving the security of the RSA system should amount to showing that the function $f : P_A \mapsto S_A$ is not computationally tractable (non-polynomial time computable). This is not known. It is clear that the function f can be reduced to either the factorization of integers or the efficient computation of the Euler ϕ-function. In fact, Miller [**?**] has shown that factorization and computing the ϕ-function are polynomial-time equivalent. But it is not known if computing f is polynomial-time equivalent to the factorization of integers. The function $\widehat{S}_A$ illustrates the concept of a "trapdoor function", which is any function that (i) is easy to compute with special knowledge (the trapdoor key, S_A), and (ii) is difficult to compute without this knowledge.
(B) Integrity of the Public Key File. We need a trusted authority to handle the public key file. A user needs to know that a pair (A, P_A) in the file (for some user A) is trustworthy. We do not want B to insert a pair (A, P_B) into the public key file, fooling us to send a message to A that B can wire-tap and read! In some applications, we need some kind of identity authentication or identify registration. E.g., A is just a name (string identifier) in the Public Key File, and in mass market applications, it would be very confusing if A can be accidentally confused with another well-known identity. For instance, in a listing of companies, we expect a company named "IBM" to be nothing else but the Big Blue. Of course, this problem is already solved by the concept of trade-marks.

Exercises

Exercise 5.1: Generalize the argument that $(M^{ed} \mathbf{mod}\, n) = M$. Suggest a cryptographic application for the choice of $n = pqr$ where p, q, r are distinct primes. ♢

Exercise 5.2: The following are consecutive prime numbers:

$$p_0 = 65,521; p_1 = 65,537; p_2 = 65,539; p_3 = 65,543; p_4 = 65,551; p_5 = 65,557$$

Note that $p_1 = 2^{16} + 1$ is the F_4 a Fermat prime. Let $N = 2^32 = 4,294,967,296$. GRADING NOTE: in the following, please organize to explicitly show all your computations, so that we only need to verify your steps.]
(a) Using these prime numbers, construct a pair (P_A, S_A) of public and private keys for Alice; similarly construct (P_B, S_B) for Bob.
(b) Compose and send the following messages using the RSA scheme and the keys constructed in part (a). You may use any appropriate protocol (A, B, AB or BA or something else). Use the simplest protocol possible. If a particular task is impossible to solve, then argue why. Recall that you may need to exploit the ability of plaintext to "self-authenticate". If the

- General Alice wants to send a message to all her troups to lay down their arms.
- Informer Alice wants to tell the FBI agent Bob that her brother stole the Rembrandt and the loot is to be found in their basement. But she wants to remain anonymous.
- Tycoon Alice is in her hideout in Bermuda and wants her Wall Street stockbroker Bob to sell all her shares of Microsoft.
- Gangster Alice wants to make a deal with arch-enemy gangster Bob: she will tell Bob where to find his favorite lost poodle if he tells Alice who killed her brother.

◊

Exercise 5.3: Class Project: implement an RSA system and send each other secret messages. To reduce the number of messages, we may divide the class into groups of 2 or 3 students. Make the following assumptions. Each pair of keys $(n, e), (n, d)$ has the property that n is more than 128 bits (but less than 130 bits). One way to ensure this is to generate $n = pq$ where p and q are each 65 bits long. We suggest that you should program in some programming language that already has a bigInteger package. Messages or plaintexts is an arbitrary ascii string (stored as a file).

(a) Write a program called CONVERT that takes such a file, and output another file that contains the original string broken up into blocks of 16 characters (the last block can be padded with blanks). Each of these blocks are interpreted as a 16-digit base 2^8 number. This number is then written out as a decimal string. The output of CONVERT is a sequence of lines, each line holding the decimal string representing a block. In general, an ascii file that contains one (arbitrary length) decimal integer per line is said to be in STANDARD FORM.

(b) Write another program UNCONVERT that takes the output of convert and reconstructs the original plaintext.

(c) Write a program CRYPT that takes a key of the form (n, e) and a file in STANDARD FORM and outputs another file in STANDARD FORM, where each integer in the input has now been encrypted using the key (n, e). Note that CRYPT can be used for encrypting as well as decrypting in the RSA system.

(d) Write a program to generate keys generate pair of keys $(n, e), (n, d)$ suitable for the RSA system (and conforming to our requirements above). To generate prime numbers, use the primality testing algorithms in the next lecture (XIX). Generate one such pair for yourself, and publish one of them for your class to read.

(e) Send to all the other groups of the class a secret message using either the AB- or BA-protocol. Inform your class instructor of your messages.

(f) When you receive ciphertexts from the other groups, decrypt them. The Fundamental Assumption of Cryptography says that you can unambiguously recover the message even though you are not told whether the AB- or BA-protocol is used. Inform your class instructor of the messages you received.

$\diamondsuit$

End Exercises

Exercises

Exercise 5.4: $\diamondsuit$

End Exercises