CS 111 Notes on Number Theory and Cryptography (Revised 1/12/2021)

1 Prerequisite Knowledge and Notation

Here are the topics on number theory that are normally covered in Math/CS11 and that you need to be familiar with (if not, review it!) in order to follow the lectures on number theory:

1. Prime and composite numbers (why there are infinitely many primes?)
2. Factorization, uniqueness (the fundamental theorem of arithmetic)
3. Relatively prime numbers
4. Greatest common divisor
5. Basic modular arithmetic and congruences

Notation for number sets. When we discuss number theory, whenever we say " a number " we mean an integer. The set of integers is denoted by $\mathbb{Z}$. Sometimes we will also say "number" when we mean that this is a natural number. This will be always clear from the context. The set of natural numbers is denoted $\mathbb{N}$. The set of positive integers is denoted $\mathbb{N}_+$. Thus

$$\begin{aligned} \mathbb{Z} &= \{..., -3, -2, -1, 0, 1, 2, 3, ...\} \\ \mathbb{N} &= \{0, 1, 2, 3, ...\} \\ \mathbb{N}_+ &= \{1, 2, 3, ...\} \end{aligned}$$

Notation for the remainder operation. For $a \in \mathbb{Z}$ and $b \in \mathbb{N}_+$, the operation of taking a remainder of a modulo b is usually denoted by $a \bmod b$ in the literature. This often leads to confusion with the attribute of the congruence relation modulo b, where we write $x \equiv y \pmod{b}$.

For the above reason, in this class we will use notation $a \operatorname{rem} b$ for the remainder operation. That is, $a \operatorname{rem} b$ is the unique integer $r \in \{0, 1, ..., b-1\}$ such that $a = q \cdot b + r$, where $q = \lfloor a/b \rfloor \in \mathbb{Z}$. Thus q is the result of the integer division of a by b and r is its *remainder* (sometimes also called the *residue*).

2 GCD and Euclid's Algorithm

Given two numbers a, b we want to compute their greatest common divisor $c = \gcd(a, b)$. This can be done using Euclid's algorithm, that is based on the following easy-to-prove theorem.

Theorem 1 *Let $a > b$. Then $\gcd(a, b) = \gcd(a - b, b)$.*

Proof: The theorem follows from the following claim: x is a common divisor of a, b if and only if x is a common divisor of $a - b, b$. To prove the claim, we show each implication separately.

($\Rightarrow$) Suppose that x is a common divisor of a and b. Then $a = \alpha x$ and $b = \beta x$ for some integers α, β. Therefore $a - b = \alpha x - \beta x = (\alpha - \beta)x$, so x is a divisor of $a - b$.

($\Leftarrow$) Suppose now that x is a common divisor of $a - b$ and b. THen $a - b = \gamma x$ and $b = \delta x$ for some integers γ, δ. This implies that $a = (a - b) + b = \gamma x + \delta x = (\gamma + \delta)x$, so x is a divisor of a. $\square$

Euclid's Algorithm. Euclid's Algorithm computes the greatest common divisor of two positive integers, and it can be written in a recursive form as follows.

function EUCLID(a, b)
  if $a = b$ **then return** a
  if $a < b$ **then** swap(a, b)
  return EUCLID$(a - b, b)$

Example. Suppose we want to compute gcd$(1034, 222)$. The algorithm will produce the following pairs of numbers: $1034, 222$; $810, 222$; $588, 222$; $366, 222$; $222, 144$; $144, 78$; $78, 66$; $66, 12$; $54, 12$; $42, 12$; $30, 12$; $24, 12$; $12, 12$. So the algorithm will return 12.

Faster version of Euclid's Algorithm. As you can see from the example above, if a is much bigger than b then the algorithm will replace a by $a-b$, $a-2b$, $a-3b$, ... For large numbers, say $a = 10^{100}$ and $b = 13$, this process would take forever. But that's easy to get around: the final result of these subtractions will be $a \operatorname{rem} b$, so instead of repeated multiplications we can replace a by $a \operatorname{rem} b$ in one step. (This will require to change the base case of recursion, replacing $a = b$ by $b = 0$ – a minor detail.) This new algorithm is very fast – it runs in time $O(\log a + \log b)$, that is the number of iterations does not exceed the total number of bits in a and b.

GCD as a linear combination. We now show that the greatest common divisor of two numbers can be expressed as their linear combination, and we adapt Eucli'd algorithm to determine this combination. This will be very useful in modular arithmetic for computing modular inverses (that you will learn about soon).

Theorem 2 *If a, b are positive integers then there exist integers α andβ such that* $\gcd(a, b) = \alpha a + \beta b$. *Further, α and β can be chosen so that $\alpha \in \{0, 1, ..., b - 1\}$.*

Proof: To prove this theorem we modify Euclid's Algorithm so that it actually computes these numbers α, β, as follows:

function EXTEUCLID(a, b)
  if $a = b$ **then return** $(a, 0, 1)$
  if $a > b$ **then**
    $(d, \alpha', \beta') =$ EXTEUCLID$(a - b, b)$
    return $(d, \alpha', \beta' - \alpha')$
  else
    $(d, \alpha', \beta') = \gcd(b - a, a)$
    return $(d, \beta' - \alpha', \alpha')$

It remains to prove that this algorithm is correct.

The proof is by induction by the number of steps of the algorithm. The base case is when there are no recursive calls, that is when $a = b$. In this case the algorithm returns $\alpha = 0$ and $\beta = 1$. Since $0 \cdot a + 1 \cdot b = b = \gcd(a, b)$, the computed values are correct.

Suppose now that the we make some number n of recursive calls in the algorithm, and assume that the theorem holds for pairs of numbers which required at most $n - 1$ recursive calls. If the inputs are a, b, where $a > b$ the algorithm will compute (d, α', β'), which by the inductive assumption are correct, that is $\gcd(a - b, b) = d$ and $\alpha' \cdot (a - b) + \beta' \cdot b = d$. The algorithm will then return $\alpha = \alpha'$ and $\beta = \beta' - \alpha'$. But then $\gcd(a, b) = d$ as well and $d = \alpha' \cdot (a - b) + \beta' \cdot b = \alpha \cdot a + \beta \cdot b$, so α and β are computed correctly. The case when $a < b$ is similar.

The argument above shows the existence of α and β in the first part of the theorem, that is $\alpha a + \beta b = d$, for $d = \gcd(a, b)$. To prove the second part, take $\alpha^* = \alpha \operatorname{rem} b$. This means that $\alpha^* = \alpha + \gamma b$, for some integer γ. Let $\beta^* = \beta - \gamma a$. Then $\alpha^* a + \beta^* b = (\alpha + \gamma b)a + (\beta - \gamma a)b = \alpha a + \beta b = d$, proving the second part of the theorem. □

In fact, the Extended Euclid's Algorithm can be modified in a similar way to compute the coefficients α and β. Therefore these coefficients can be computed in polynomial time. (This is important for what we will do later.)

In hand calculations we will typically not use Extended Euclid's Algorithm to find the values of α and β. Instead, for small numbers a, b, one can simply list all multiples a and b, until finding two whose difference is $c = \gcd(a, b)$.

Example. Consider $a = 22$ and $b = 32$. Then $\gcd(a, b) = 2$, so we want to find their multiples that differ by 2. The list of their multiples is

$$\begin{array}{cccc} 0 & 22 & 44 & 66 \\ 0 & 32 & 64 & \end{array}$$

so we get $3 \cdot 22 - 2 \cdot 32 = 2$. Thus $\alpha = 3$ and $\beta = -2$.

3 Modular Arithmetic

Congruence relation. Recall that notation "$\equiv$" represents congruence relations on integers. For integers $x, y \in \mathbb{Z}$ and a positive integer $k \in \mathbb{N}_+$, we write $x \equiv y \pmod{k}$ iff $k \mid x - y$. For any k, this congruence relation is an equivalence relation on the set of all integers $\mathbb{Z}$. Therefore it partitions $\mathbb{Z}$ into equivalence classes. By $[x]$ we denote the equivalence class of x. In this context, we call x the *representative* of class of $[x]$. The choice of a representative is not unique; in fact each class has infinitely many representatives. For instance, $[x] = [x + k] = [x + 2k] = ...$

Example. Let $k = 5$. Then the equivalence classes of the congruence relation modulo 5 are:

$$\begin{aligned} [0] &= \{... - 10, -5, 0, 5, 10, ...\} \\ [1] &= \{... - 9, -4, 1, 6, 11, ...\} \\ [2] &= \{... - 8, -3, 2, 7, 12, ...\} \\ [3] &= \{... - 7, -2, 3, 8, 13, ...\} \\ [4] &= \{... - 6, -1, 4, 9, 14, ...\} \end{aligned}$$

As explained earlier, the choice of representative is not unique. As an example, any number from equivalence class [2] can be chose as its representative; that is $[2] = [-3] = [7]$, etc.

An important property of equivalence relations is that when we perform arithmetic operations modulo k, then the result is independent of which member from a congruence class we use. The theorem below captures this property.

Theorem 3 *Suppose that* $x \equiv y \pmod{k}$ *and* $u \equiv v \pmod{k}$. *Then*

- $(x + u) \equiv (y + v) \pmod{k}$, *and*
- $(xu) \equiv (yv) \pmod{k}$.

Example. Let $k = 5$ again. Then $(-4) \equiv 11 \pmod{5}$ and $9 \equiv 24 \pmod{5}$. We then have $(-4) + 9 = 5$ and $11 + 24 = 35$, and $(-4 + 9) \equiv (11 + 24) \pmod{5}$. Similarly, $(-4) \cdot 9 = -36$ and $11 \cdot 24 = 264$. Since $264 - (-36) = 300$, we have $((-4) \cdot 9) \equiv (11 \cdot 24) \pmod{5}$.

Modular arithmetic. Modular arithmetic involves performing algebraic operations on integers, with all operations taken modulo some fixed parameter k. Typically, our objective is to compute the value of some expression modulo k. The operations we consider are addition, multiplication, and exponentiation (where the exponent is an arbitrary natural number).

As an example, we may want to compute $7 \cdot 3^6 \operatorname{rem} 5$. This can be calculated as follows:

$$(7 \cdot 3^9) \operatorname{rem} 5 \;=\; (7 \cdot 9683) \operatorname{rem} 5 \;=\; 137781 \operatorname{rem} 5 \;=\; 1.$$

Such a computation could be tedious when done by hand. But, according to Theorem 3, whenever we compute modular arithmetic and have a number x in the formula, we can replace it by any x' that is

congruent to x modulo 5, that is $x \equiv x' \pmod 5$. Another way to think about this is that the "numbers" on which we performm the calculation are in fact whole equivalence classes of relation $\equiv \pmod 5$.

So instead of the above, we can perform this computation without any need for a calculator, as follows:

$$\begin{aligned} 7 \cdot 3^9 &\equiv 2 \cdot 3(3^8) \pmod 5 \\ &\equiv 2 \cdot 3 \cdot (3^8) \pmod 5 \\ &\equiv 6 \cdot (3^2)^4 \pmod 5 \\ &\equiv 1 \cdot (9)^4 \pmod 5 \\ &\equiv (-1)^4 \pmod 5 \\ &\equiv 1 \pmod 5 , \end{aligned}$$

which gives us that $(7 \cdot 3^9) \operatorname{rem} 5 = 1$.

Computing powers. Suppose that we are asked to compute $3^{17} \operatorname{rem} 7$, without using a calculator. This seems daunting, as 3^{17} is a huge number. But there is a trick to do this faster, by replacing multiplications by squaring: When the exponent is even, we divide the exponent by 2 and square the argument. When the exponend is odd, we factor out one argument out and apply the squaring to the remaining exponent. More specifically, at each step we can apply the following rule, when $m \geq 2$:

$$x^m = \begin{cases} (x^2)^{m/2} & \text{if } m \text{ is even} \\ x(x^2)^{(m-1)/2} & \text{if } m \text{ is odd} \end{cases}$$

And at each step we can replace the argument by its remainder (or any number congruent to it). In this example, this will work as follows:

$$\begin{aligned} 3^{17} &\equiv 3 \cdot 3^{16} \pmod 7 \\ &\equiv 3 \cdot 9^8 \pmod 7 \\ &\equiv 3 \cdot 2^8 \pmod 7 \\ &\equiv 3 \cdot 4^4 \pmod 7 \\ &\equiv 3 \cdot 16^2 \pmod 7 \\ &\equiv 3 \cdot 2^2 \pmod 7 \\ &\equiv 3 \cdot 4 \pmod 7 \\ &\equiv 12 \pmod 7 \\ &\equiv 5 \pmod 7 , \end{aligned}$$

so $3^{17} \operatorname{rem} 7 = 5$.

3.1 Modular Inverses

As discussed in CS11 and earlier in this class, for some number n we can consider numbers $0, 1, ..., n-1$ as a small "universe" and redefine the operations of addition, subtraction and multiplication by taking the result modulo n, so that the resulting value is in the set $0, 1, ..., n-1$. This way this set is closed under these operations.

We can also compute powers. For example, what is $3^{44208} \operatorname{rem} 8$? We can write $3^{44208} = (3^2)^{22104} = 9^{22104}$, so $3^{44208} \operatorname{rem} 8 = 9^{22104} \operatorname{rem} 8 = 1^{22104} \operatorname{rem} 8 = 1$. As this example illustrates, when you do computation in a modular arithmetic, you can compute the remainders at each step, rather than computing the complete value first and then taking the remainder.

We would like to extend this even further. In real numbers, we can also divide x/y. Can we do the division in our modular arithmetic? As it turns out, the answer is yes if n is a prime. (This can be generalized to all numbers n, but we will not cover it in this class.)

So let us consider a prime p and numbers $1, 2, ..., p-1$. We do not include 0 because division by 0 is not allowed, even for real numbers. In real arithmetic, $x/y = x \cdot y^{-1}$, and we could do the same for modular arithmetic, but then we need to figure out what is y^{-1}, the inverse of y. This is easy: Define $y^{-1} \pmod p = z$, for $z \in \{1, 2, ..., p-1\}$, iff $yz = 1 \pmod p$. We now show that this inverse exists and is uniquely defined.

Theorem 4 *Let p be a prime and $y \in \{1, 2, ..., p-1\}$. Then the inverse of y modulo p exists and is unique (among numbers between 1 and $p-1$.)*

Proof: We show existence first. Since y and p are relatively prime, there are α, β such that $\alpha y + \beta p = 1$. Taking the remainder modulo p of both sides, we get $\alpha y = 1 \pmod p$. So α is the inverse of y. This is almost right, but not exactly, because α may not be in the range $1, 2, ..., p-1$. If so, replace α by $\alpha' + dp$, where d is chosen so that $\alpha' \in \{1, 2, ..., p-1\}$.

To show uniqueness, towards contradiction, suppose there are two different inverses z, z' of y between 1 and $p-1$. So $zy = 1 \pmod p$ and $z'y = 1 \pmod p$. Thus $y(z - z') = 0 \pmod p$, and since $\gcd(y, p) = 1$, we conclude that $z = z' \pmod p$, so z, z' must be equal. □

Example. Find $10^{-1} \pmod{13}$. We first find α and β for which $\alpha \cdot 10 + \beta \cdot 13 = 1$. We list the multiples of 10 and 13:

0	10	20	30	40
0	13	26	39	

so we get $4 \cdot 10 - 3 \cdot 13 = 1$. Thus $4 \cdot 10 = 1 \pmod{13}$, so $10^{-1} = 4 \pmod{13}$.

Linear congruences. A linear congruence is an equation of the form

$$ax \equiv b \pmod p.$$

We will conisder only the case when p is a prime number (for now). It's essentially an equation that we want to solve for x (where a, b are given.) Is it always possible? If so, is the solution unique?

Example: Let's say we want to solve $7x \equiv 5 \pmod 9$. Determine first the inverse of 7 modulo 9, that is, we want to find z for which $7z \equiv 1 \pmod 9$. This is equivalent to finding z and s for which $7z = 9s + 1$. So we search for a multiple of 7 among the numbers of the form $9s + 1$: 10, 19, 28, 37, ... The first number in this sequence that will work is 28, since it's a multiple of 7. Since $7 \cdot 4 = 28$, we get $z = 4$. Indeed, $7 \cdot 4 \equiv 1 \pmod 9$.

Now, to solve the original congruence, multiply it by 4, and we get $x = 20 \pmod 9 = 2$.

Fermat's Theorem

In this lecture we focus again on the properties of integers modulo a prime. Let p be a prime and consider numbers $0, 1, ..., p-1$, namely all possible remainders modulo p. As we discussed earlier, we can think of this set as an algebraic system, with arithmetic operations of addition and multiplication modulo p. For example, for $p = 7$, if we compute all results modulo 7, we have $4 + 5 = 2$, $4 \cdot 5 = 6$, etc. We have also discussed inverses modulo p. For example, the inverse of 5 modulo 7 is 3, because $5 \cdot 3 \equiv 1 \pmod 7$.

We can also consider computing powers, $a^x \operatorname{rem} p$, for $a \in \{1, 2, ..., p-1\}$ and any non-negative integer x. This raises some interesting questions. If we compute the sequence of numbers $a^0, a^1, a^2, ...$, all taken modulo p, this gives us an infinite sequence with all numbers in the finite range $\{1, 2, ..., p-1\}$. Will this sequence visit all numbers in this range? Will this sequence have a pattern? In particular, will it be periodic?

As it turns out, yes, if p is prime then this sequence will be periodic, namely it will repeat itself every $p-1$ steps, and each block of $p-1$ consecutive elements of this sequence is a permutation of $\{1, 2, ..., p-1\}$. This follows from the following theorem.

Theorem 5 *(Fermat's Little Theorem) Let p be a prime number and let $a \in \{1, 2, ..., p-1\}$. Then $a^{p-1} \equiv 1 \pmod{p}$.*

Proof: Fix our a, and consider the sequence of the numbers

$$a \cdot 1\,,\ a \cdot 2\,,\ \ldots\,,\ a \cdot (p-1) \tag{1}$$

computed modulo p.

We claim that all these numbers are different. The proof of this claim is by contradiction. Suppose that there are two different $x, y \in \{1, 2, ..., p-1\}$, for which $ax \equiv ay \pmod{p}$. Without loss of generality, assume $x > y$. (Otherwise we can swap x and y.) This gives us that $a(x-y) \equiv 0 \pmod{p}$, which is equivalent to saying that $a(x-y) = kp$ for some integer p. On the left-hand side we have an integer that is a product of two numbers, a and $x-y$, both smaller than p, so it does not have p in its factorization. But on the right-hand side, we have an integer that has p in its factorization, so this equation cannot be true, giving us our contradiction.

So now we know that all numbers in the sequence (1) are different (remember that we compute them modulo p). This sequence has $p-1$ different numbers, all in the range $1, 2, ..., p-1$, so the sequence must be simply a permutation of $1, 2, ..., p-1$. Therefore if we multiply these numbers, we get

$$(a \cdot 1) \cdot (a \cdot 2)\,,\ \ldots\,,\ \cdot(a \cdot (p-1)) \equiv 1 \cdot 2 \cdot \ldots \cdot (p-1) \pmod{p}$$

We can cancel all numbers $1, 2, ..., p-1$ from this equation (multiplying both sides by their inverses), which leaves us with

$$a^{p-1} \equiv 1 \pmod{p},$$

completing the proof of the theorem. □

Example 1. Consider $p = 7$ and $a = 3$. Then Fermat's theorem says that $3^{7-1} = 3^6 \equiv 1 \pmod{7}$. Indeed, computing modulo 7, $3^6 = 9^3 = 2^3 = 8 = 1$.

Fermat's Theorem has some interesting applications, the most notable of which is for the correctness of the RSA (that we will discuss later). We can also use it to compute inverses and powers, as explained below.

Application to computing inverses. The statement of Fermat's theorem suggests that it should be useful to computing inverses. We have $a^{p-1} \equiv 1 \pmod{p}$. But we can write a^{p-1} as $a \cdot a^{p-2}$. This gives us that

$$a \cdot a^{p-2} \equiv 1 \pmod{p}$$

But this implies that $a^{p-2} \equiv a^{-1} \pmod{p}$, that is $a^{p-2} \operatorname{rem} p$ is exactly the inverse of a modulo p.

For example, to compute $3^{-1} \pmod{7}$, we can compute $3^5 \operatorname{rem} 7$. Computing modulo 7, this gives us $3^{-1} = 3^5 = 3 \cdot 9^2 = 3 \cdot 2^2 = 3 \cdot 4 = 12 = 5$. This is indeed correct, because $3 \cdot 5 \equiv 1 \pmod{7}$.

Application to computing powers. Let p be prime and $a \in \{1, 2, ..., p-1\}$. Suppose we want to compute $a^x \operatorname{rem} p$, for some large x. We can do this using the squaring algorithm, but Fermat's theorem could also significantly reduce the computation. Let $z = x \operatorname{rem} (p-1)$, and write x as $x = r(p-1) + z$. Then, computing modulo p, we have

$$a^x = a^{r(p-1)+z} = (a^{p-1})^r \cdot a^z = a^z.$$

Since z is much smaller than x, this can save us a lot of computation.

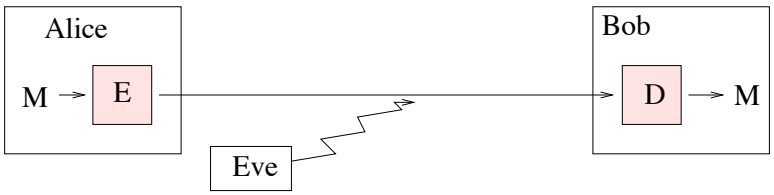
For example, to compute $3^{40954327098} \operatorname{rem} 11$, we can proceed as follows (all calculations modulo 11):

$$\begin{aligned} 3^{40954327098} &= 3^{10 \cdot 4095432709} \cdot 3^8 \\ &= 3^8 = 9^4 = 81^2 = 4^6 = 16 = 5. \end{aligned}$$

Just to make sure though: this trick is legal only if p is prime.

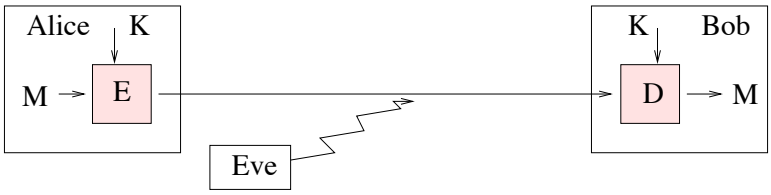
The RSA Cryptosystem

Some cryptography basics. Before we get to the RSA, we need to understand what is public-key encryption and why it is needed. The general setting is this: we have two agents, called Alice and Bob, and Alice wants to send some private information to Bob over a public channel. In order to hide its content, she scrambles her message using some encryption algorithm. Upon receiving the encrypted message, Bob will use the appropriate decryption algorithm to recover the original message. The cryptographers sometimes introduce another party, called Eve (eavesdropper), that listens to the public channel and tries to figure out what Alice and Bob are saying to each other.



Secret algorithms. One approach is to use a *secret encryption algorithm* E, that Alice applies to her message, while Bob uses the corresponding secret decryption algorithm D that he applies to the received ciphertext. In order to recover the original message, E and D must have the property that $D(E(M)) = M$ for each M. Thus, D is the inverse of E, $D = E^{-1}$. This method has two drawbacks. First, it is completely infeasible in a large system like, for example, Internet. How do Alice and Bob agree on a common algorithm? Most parties that want to exchange private messages do not have time, desire, nor sufficient knowledge of cryptography to design their own secret algorithms. In fact, in practice they do not even know each other. Another issue is security. Somewhat paradoxically, a secret-algorithm system may well be less secure than an algorithm with a public algorithm, because it has not been properly verified against possible attacks.

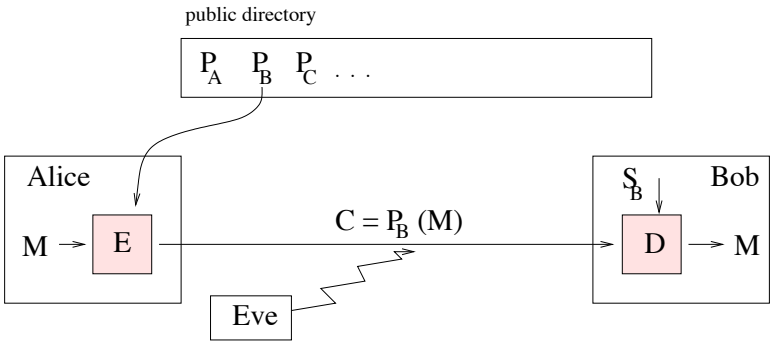
Public algorithm with secret key. These concerns can be addressed by using a *public encryption/decryption algorithm* that uses a *secret-key*. The encryption has two arguments: a message M and a secret key K. In general, for different keys the encrypted message $C = E_K(M)$ will be different. Bob knows the sercret key and uses it to decode the message, $M = D_K(C)$. Of course, as before, we need that for each key K, D_K is the inverse of E_K.



The main advantage of this system is that now the users can use standard, well tested, algorithms. Historically, one commonly used standard was DES (Data Encryption Standard) set in 1977 by the National Bureau of Standards. DES used 56-bit keys (plus 8 parity bits) for enciphering 64-bit blocks of data. DES consists of a sequence of rather involved transformations that can be very efficiently implemented in hardware. Longer messages are broken into 64-bit blocks. A naive approach, called the ECB mode (Electronic Code-Book), is to encipher each block separately. This method is not safe and is not recommended. Instead, one can use techniques that use feedback to make the encoding of each block depend on all previous blocks. DES is now outdated because, due to improvements in hardware performance, its 56-bit keys are not secure anymore. Appropriate modifications of DES (for example, triple-DES) are now commonly used, for example by banks (ATMs, inter-bank transactions). Other currently used block ciphers are IDEA (used in PGP) and Blowfish.

One problem still remains though: How do Alice and Bob agree on what key to use? We get into a chicken-and-egg problem here: In order to exchange secret messages over a public channel, they need to agree on a secret key, but this needs to be done over a public channel too! Sounds impossible, huh. As we shall soon see, it can be done. There are protocols for key exchange over a public channel. Another option is to use a public-key system. Read on.

Public-key systems. The trick behind public-key systems is to have *two keys* for each user, one public and one secret. The public key is used for encrypting the messages sent to this user, and the secret key is used for decryption. Denote Alice's public key by P_A, and her secret key by S_A. For brevity, we will write $P_A(M)$ instead of $E_{P_A}(M)$, and $S_A(C)$ instead of $D_{S_A}(C)$. As before, the encryption and decryption algorithms need to be the inverses of each other, that is $S_A(P_A(M)) = M$.



In order for this approach to work, we need to make sure that the secret keys cannot be computed from the public keys. What is exactly meant by "cannot"? After all, one could just try all possible keys and see if one of them works. However, when the keys are large enough, say 200 digits long or so, this brute-force approach is not feasible. The key length (or, in general, the strength of the system) should be such that the (expected) cost of computing the secret key substantially exceeds potential benefits of breaking the system.

The idea of public-key systems was introduced by Diffie and Hellman in 1976. It has several implementations. One, developed by Merkle and Hellman (and patented) was based on the knapsack problem. Their algorithm was subsequently broken by Shamir and others. Other methods include the RSA and ElGamal systems.

The RSA cryptosystem. The RSA (Rivest-Shamir-Adleman) is the most popular implementation of public-key systems currently in use. Although not *provably* secure, significant effort has been devoted to finding some way to break this system, and these have not been successful.

In RSA, we think of each message M as a number of some fixed length. If the message is longer, it is broken into smaller pieces. We describe the system in three parts: initialization, encryption, and decryption.

Initialization: Bob picks $n = pq$, where p, q are two large distinct primes, say 100 digits long. Then he selects a small integer e relatively prime to $\phi(n) = (p-1)(q-1)$. Let $d = e^{-1} \operatorname{rem} \phi(n)$, the inverse of e modulo $\phi(n)$. He publishes $P_B = (e, n)$ as the public key and hides $S_B = d$, his secret key.

Encryption: Each message is represented by an integer M between 0 and $n-1$. If Alice wants to send M to Bob, she computes $C = P_B(M) = M^e \operatorname{rem} n$, and sends C to Bob.

Decryption: Upon receiving a ciphertext C, Bob computes $S_B(C) = C^d \operatorname{rem} n$. The value of $S_B(C)$ is the decrypted message.

Theorem 6 *RSA is correct, that is $S_B(P_B(M)) = M$ for all messages $M \in \{0, 1, ..., n\}$.*

Proof: To prove that $S_B(P_B(M)) = M$, we need to show that $M^{ed} = M \pmod{n}$. Recall that $\phi(n) = (p-1)(q-1)$, because $n = pq$. Since $e = d^{-1} \pmod{\phi(n)}$, we have

$$ed \;=\; 1 + k(p-1)(q-1),$$

for some k and, therefore, if $M \neq 0 \pmod{p}$, by Fermat's Little Theorem we get:

$$M^{ed} \;=\; M(M^{p-1})^{k(q-1)} \;=\; M \pmod{p}$$

If $M = 0 \pmod{p}$ then $M^{ed} = M \pmod{p}$ as well. Similarly we get $M^{ed} = M \pmod{q}$. Thus $M^{ed} - M$ is a multiple of both p and q. Since p, q are prime, $M^{ed} - M$ must be also a multiple of $pq = n$. This implies that $M^{ed} = M \pmod{n}$, and we are done. □

Example. Suppose that Bob picks $p = 7$ and $q = 11$. Then $n = pq = 77$, and $\phi(n) = (p-1)(q-1) = 60$. Bob picks $e = 13$, and $d = 13^{-1} = 37 \pmod{60}$. Bob's public key is $P_B = (13, 77)$ and his secret key is $S_B = (37, 77)$. The encryption and decryption algorithms are:

$$P_B(M) \;=\; M^{13} \operatorname{rem} 77 \qquad\qquad S_B(C) \;=\; C^{37} \operatorname{rem} 77$$

Now, let's suppose that Alice wants to send a message $M = 5$ to Bob. She sends him

$$\begin{aligned} C \;=\; P_B(5) \;&=\; 5^{13} \operatorname{rem} 77 \\ &=\; 5 \cdot ((5 \cdot 5^2)^2)^2 \operatorname{rem} 77 \\ &=\; 5 \cdot ((48)^2)^2 \operatorname{rem} 77 \\ &=\; 5 \cdot (71)^2 \operatorname{rem} 77 \\ &=\; 5 \cdot 36 \operatorname{rem} 77 \\ &=\; 26 \end{aligned}$$

Bob receives $C = 26$ and decodes it using his secret key:

$$\begin{aligned} M \;=\; S_B(26) \;&=\; 26^{37} \operatorname{rem} 77 \\ &=\; 26 \cdot ((26 \cdot ((26^2)^2)^2)^2)^2 \operatorname{rem} 77 \\ &=\; \cdots \\ &=\; 5 \end{aligned}$$

Security. We need to decide first what exactly we mean by "security", or, what does it mean to "break the system". For our purpose, assume that by breaking the cryptosystem we mean computing the secret key S_A from the public key P_A. One way to do it is by exhaustive search: Given P_A, generate some pair (M, C), where M is a message and $C = P_A(M)$ is its encryption, and then try all keys S, and for each compute $M' = S(C)$, until $M' = M$. Then $S_A = S$. Of course, this brute-force approach is impractical for large keys.

Ideally, we would like to formally *prove* that our cryptosystem cannot be broken, that is, that there is no polynomial-time algorithm that computes the secret key from the public key. Unfortunately, noone knows how to prove such impossibility results for natural computational problems. The next best thing would be to show the following: if we could break the cryptosystem, then we could also solve some other computational problem which is believed to be hard. One such notoriously hard problem is factorization.

Notice that if you can factor fast, you can break the RSA: Given n, compute its factorization $n = pq$. Then you can compute $\phi(n) = (p-1)(q-1)$, and finally, you compute $d = e^{-1} \operatorname{rem} \phi(n)$ using Extended Euclid's algorithm, and we're done. This does not prove, however, that breaking the RSA is as hard as factoring, since there could be other ways of computing $\phi(n)$ that do not use factorization, or there could be a way to compute d without computing $\phi(n)$. Nevertheless, in spite of intensive research in this area, no such method has been yet discovered.

Efficiency. Of course, for the cryptosystem to be useful, it's important that all computation in the initialization, encryption and decryption steps be efficient. This is not an algorithm's class, so we will only briefly address this issue.

In the initialization stage, we need to first find large primes. This can be done by simply guessing large numbers and verifying their primality. The Prime Number Theorem implies that we only need to make a small number of guesses, in expectation, and primality can be verified efficiently. We also need to compute an inverse of e modulo $\phi(n)$. This can be done using the Extended Euclid's Algorithm.

The encryption/decryption stages involve computing large powers modulo n. The public exponent e is often small, but the private exponent d could be the same order of magnitude as n, that is 2048 bits or more. If we simply multiplied C by itself C times (each time taking the remainder modulo n), this could involve some 2^{200} multiplications, which is hopeless.

But there is a better way, called the squaring algorithm. Suppose you want to calculate $3^{32} \operatorname{rem} 7$. You can write it as $((((3^2)^2)^2)^2)^2 \operatorname{rem} 7$, so instead of 31 multiplications, this computation will only require 5 multiplications. If the exponent is not a power of 2, this can be slightly modified, by factoring 3 in this process whenever the current exponent is odd. For example, $3^{35} \operatorname{rem} 7 = 3 \cdot (3 \cdot (((3^2)^2)^2)^2)^2 \operatorname{rem} 7$.