Cryptography

Rachid Hamadi, CSE, UNSW

COMP9021 Principles of Programming, Term 3, 2019

[1]:
```
from random import randrange
```

Prime numbers and some number theoretic results play a crucial role in the production and use of safe secret keys. We start with a presentation of some essential number theoretic concepts and results, that will then be applied to two fundamental cryptographic protocols to let two agents, Alice and Bob, communicate safely, despite of the fact that they are far away from each other and the information that they exchange over a network can be intercepted.

1 Number theoretic background and extended Euclidean algorithm

Bézout's identity is the following statement:

Proposition 1. *Let nonzero integers a and b be given. There exists integers x and y with $ax + by = \gcd(a, b)$.*

Proof. Let c be a nonzero integer of the form $ax + by$ with least absolute value. By changing the signs of x and y, we can assume that c is positive. Of course, $c \leq \min(|a|, |b|)$. Then c divides a, as otherwise a would be of the form $ck + r$ with $0 < r < c$, hence $r = a - ck = a - (ax + by)k = a(1 - x) + b(-yk)$, which contradicts the definition of c. Similarly, c divides b. Also, if c' divides both a and b, then c' divides c, completing the verification that $c = \gcd(a, b)$. □

Given nonzero natural numbers a and b, the computation of $\gcd(a, b)$ and two natural numbers x and y with $\gcd(a, b) = ax + by$ can be achieved thanks to the *extended Euclidean algorithm*. Given $(a, 0)$ as input, the extended Euclidean algorithm returns $(a, 1, 0)$, which is correct since $a = a \times 1 + 0 \times 0$ and $a = \gcd(a, 0)$. Given (a, b) with $b \neq 0$ as input, the extended Euclidean algorithm applies itself to $(b, a \bmod b)$, which yields a triple of the form (c, x, y), and returns $(c, y, x - \lfloor a/b \rfloor y)$. So the extended Euclidean algorithm generalises the Euclidean algorithm, which for all natural numbers a and b with $b \neq 0$, computes $\gcd(a, b)$ as $\gcd(b, a \bmod b)$ (Note that in case $b > a$, these algorithms swap both arguments, and then the first argument is always greater than the second one in all recursive calls.) The Euclidean algorithm is correct because a divisor of b is a divisor of a number of the form $bk + r$ iff it is a divisor of r. The extended Euclidean algorithm is correct because of the following sequence of equalities:

$$xb + y(a \bmod b) = xb + y(a - \lfloor a/b \rfloor b) = ya + (x - \lfloor a/b \rfloor y)b$$

Given nonzero integers a and b, the computation of $\gcd(a, b)$ and two natural numbers x and y with $\gcd(a, b) = ax + by$ can be done by computing as described $\gcd(|a|, |b|)$ and two integers x and y with $\gcd(|a|, |b|) = |a|x + |b|y$, changing the sign of x if $a < 0$, and changing the sign of y if $b < 0$.

Two integers are *coprime*, or one of them is *coprime to* the other, if their gcd (greatest common positive divisor) is 1. Note that 0 is coprime to 1, to -1, and to no other integer. Note that 1 is coprime to all integers.

Given an integer p, $\Phi(p)$ denotes the set of strictly positive integers at most equal to $|p|$ that are coprime to p. For instance, $\Phi(0) = \varnothing$, $\Phi(1) = \{1\}$, $\Phi(6) = \{1,5\}$, and $\Phi(15) = \{1,2,4,7,8,11,13\}$. The *totient* of an integer p, denoted $\phi(p)$, is the number of elements in $\Phi(p)$. For instance, $\phi(6) = 2$ and $\phi(15) = 7$. In particular, if p is a prime natural number then $\Phi(p) = \{1,\ldots,p-1\}$ and $\phi(p) = p-1$.

From now on, p denotes an integer greater than 1.

Proposition 2. *Let an integer a be given. Then a has an inverse modulo p iff a is coprime to p; if a has an inverse modulo p then there exists $b \in \Phi(p)$ such that for all integers c, c is an inverse of a iff $c \bmod p = b$.*

Proof. An integer b is an inverse of a modulo p iff $ab = 1$ modulo p, which is equivalent to the existence of an integer y such that $ab + py = 1$. If a is coprime to p then the existence of integers b and y satisfying $ab + py = 1$ follows from Proposition (1). Conversely, suppose that integers b and y are such that $ab + py = 1$. If c is a strictly positive integer that divides both a and p, then c divides $ab + py$, hence c divides 1, hence c is equal to 1. Therefore, a is coprime to p. So we have verified that a has an inverse modulo p iff a is coprime to p. If a has an inverse b modulo p then a is an inverse of b modulo p, and so $b \in \Phi(p)$ by what has just been established; also, every integer equal to b modulo p is an inverse of a modulo p. Suppose that a has two inverses b and c modulo p. Then we have the following equalities modulo p: $b = b.1 = b(ac) = (ab)c = 1.c = c$. Hence b and c are equal modulo p, completing the proof of the proposition. □

Proposition 3. *Let a be an integer that is coprime to p. There exists a unique positive integer N such that for all integers b that are coprime to p, $ba^0 \bmod p$, $ba^1 \bmod p$, ..., $ba^N \bmod p$ are pairwise distinct and ba^N is equal to b modulo p (in particular, $a^N \bmod p = 1$).*

Proof. The result is straightforward if $a \bmod p = 1$ (N is then equal to 0), so suppose otherwise. Since $\{a^i \bmod p \mid i \in \mathbf{N}\}$ is finite, there exists integers i and j with $i < j$ such that $a^i \bmod p = a^j \bmod p$, so multiplying both sides of the equality by the inverse of a modulo p, which exists by Proposition (2), yields the equality $a^{j-i} \bmod p = 1$. Choose such i and j so that $j - i$ is minimal; set $N = i - j$. Let an integer b that is coprime to p be given. Then ba^N is equal to b modulo p. Assume for a contradiction that there exists a positive integer k with $k < N$ and $ba^k \bmod p = b \bmod p$. Then applying Proposition (2) again and multiplying both sides of the equality by the inverse of b modulo p yields $a^k \bmod p = 1$, which together with the fact that k cannot be equal to 0 by the assumption on a, contradicts the definition of N. □

Given an integer a that is coprime to p, the *cycle length of a for p* is defined as the integer N whose existence and uniqueness is stated in Proposition (3).

Euler's theorem states:

Theorem 4. *Let a be an integer that is coprime to p. Then $a^{\phi(p)} \bmod p = 1$. In particular, if p is prime then $a^{p-1} \bmod p = 1$ (Fermat's little theorem).*

Proof. Let N be the cycle length of a for p. By Proposition (3), $\left\{\{ba^i \bmod p | i \in \mathbf{N}\} \mid b \in \Phi(p)\right\}$ is a partition of $\Phi(p)$ into sets of the same cardinality N; let M be the number of members of the partition, so $NM = \phi(p)$. Then $a^{\phi(p)} \bmod p = (a^N)^M \bmod p = 1^M \bmod p = 1$. □

The Chinese Remainder Theorem states:

Theorem 5. *Let d and e be coprime strictly positive integers. There is a unique one-to-one correspondence between the set of integers n with $0 \leq n < de$ and the set of pairs of integers (a, b) with $0 \leq a < d$ and $0 \leq b < e$ such that for all integers x, $x \bmod de = n$ iff $x \bmod d = a$ and $x \bmod e = b$.*

Proof. A mapping with the stated property must map every integer n with $0 \leq n < de$ to the (unique) pair (a, b) of integers such that $n \bmod d = a$ and $n \bmod e = b$. Suppose that there are two integers n and m with $0 \leq n, m < de$ that are mapped to the same pair (a, b). Then $m - n$ would be a multiple of both d and e, and since d and e are coprime, a multiple of de. This together with the fact that $0 \leq n, m < de$ implies that $m = n$. Hence the mapping is one-to-one indeed. Let integers a, b and n with $0 \leq n < de$ be such that n is mapped to (a, b). Write n as $a + ud$ and as $b + ve$ for some integers u and v. Then for all integers x, $x \bmod de = n$ iff $x = n + kde$ for some integer k, iff $x = a + ud + kde$ and $x = b + ve + kde$ for some integer k, iff $x \bmod d = a$ and $x \bmod e = b$. □

Proposition 6. *Let d and e be coprime strictly positive integers. Then $\phi(de) = \phi(d)\phi(e)$.*

Proof. The proposition is straightforward if $d = 1$ or $e = 1$, so suppose otherwise. Consider the one-to-one mapping M from the set of integers n with $0 \leq n < de$ to the set of pairs of integers (a, b) with $0 \leq a < d$ and $0 \leq b < e$ with the property stated in Theorem (5). Note that M maps 1 to $(1, 1)$. Let an integer x with $0 \leq x < de$ be given. By that theorem, for all integers x, $xy \bmod de = 1$ iff $x \bmod d = 1$ and $y \bmod e = 1$. So by Proposition (2), $x \in \Phi(de)$ iff $x \in \Phi(d)$ and $x \in \Phi(e)$. Hence the image of $\Phi(de)$ by M is $\Phi(d) \times \Phi(e)$, implying the claim of the proposition. □

For instance, $\Phi(5) = \{1, 2, 3, 4\}$, $\Phi(6) = \{1, 5\}$, $\Phi(30) = \{1, 7, 11, 13, 17, 19, 23, 29\}$, and $\phi(30) = 8 = 4 \times 2 = \phi(5)\phi(6)$.

Proposition 7. *The sum of all numbers of the form $\phi(d)$ with d a positive divisor of p is equal to p.*

Proof. There are p pairs of the form (n, p) with $1 \leq n \leq p$. Each of these pairs maps to the unique fraction that simplifies $\frac{n}{p}$, of the form $\frac{d}{e}$ with e a positive divisor of p and d an integer with $0 < d \leq e$ that is coprime to e. Conversely, a fraction of the form $\frac{d}{e}$ with e a positive divisor of p and d an integer with $0 < d \leq e$ that is coprime to e is what the pair $(\frac{dp}{e}, p)$ maps to. □

For instance, if $p = 12$ then the divisors of p are 1, 2, 3, 4, 6 and 12, $\Phi(1) = \{0\}$, $\Phi(2) = \{1\}$, $\Phi(3) = \{1, 2\}$, $\Phi(4) = \{1, 3\}$, $\Phi(6) = \{1, 5\}$, $\Phi(12) = \{1, 5, 7, 11\}$, and $1 + 1 + 2 + 2 + 2 + 4 = 12$.

Proposition 8. *Let d be a strictly positive integer. The number of elements of $\Phi(p)$ of cycle length d for p is equal to either 0 or $\phi(d)$.*

Proof. Suppose that this number is not 0. Let $a \in \Phi(p)$ be of cycle length d for p. Every integer e with $0 \leq e < d$ satisfies $(a^e)^d \bmod p = (a^d)^e \bmod p = 1$, therefore providing a distinct root a^e modulo p of the polynomial $X^d - 1$. Since a polynomial of degree d has at most d roots, it follows that every member of $\Phi(p)$ of cycle length d is of the form a^e for some positive integer e. Let a positive integer e be given. Then for all positive integers f, $(a^e)^f \bmod p = 1$ iff ef is a multiple of d, which is equivalent to f being a multiple of $\frac{d}{\gcd(d,e)}$. Hence the cycle length of e for p is equal to $\frac{d}{\gcd(d,e)}$, which is equal to d iff $e \in \Phi(d)$, completing the proof of the proposition. □

An integer a is a *primitive root of p* if a is coprime to p and the cycle length of a for p is equal to $p - 1$.

Theorem 9. *Suppose that p is prime. Then there are $\phi(p - 1)$ primitive roots of p.*

Proof. Let a be an integer that is coprime with p. Let N be the cycle length of a for p. By Proposition (3), for all members b of $\Phi(p)$, $\{ba^i \bmod p | i \in \mathbf{N}\}$ is a subset of $\{1, \ldots, p-1\}$ of cardinality N. Hence N divides $p-1$. Let D be the set of positive divisors of $p-1$. For all $d \in D$, let N_d be the number of elements of $\{1, \ldots, p-1\}$ of cycle length d for p. By Proposition (8), for all $d \in D$, N_d is equal to either 0 or $\phi(d)$. Also, $\Sigma_{d \in D} N_d = p-1$. This together with Proposition (7) implies that for all $d \in D$, $N_d = \phi(d)$. In particular, there are $\phi(p-1)$ primitive roots of p. □

2 The Diffie-Hellman protocol

This protocol is symmetric. It lets Alice and Bob come up with a secret code thanks to which they can encode and decode messages.

Alice chooses a prime number p and a primitive root g of p. She sends p and g across the network. Alice and Bob generate secret numbers a and b. Alice computes $A = g^a \bmod p$ and Bob computes $B = g^b \bmod p$. Alice sends A to Bob, who computes $A^b \bmod p$, and Bob sends B to Alice, who computes $B^a \bmod p$. As $B^a = (g^b)^a = g^{ba} = (g^a)^b = A^b$, Alice and Bob compute the same number; it is their secret key that they can use to encode and decode their messages in one way or another.

Someone eavesdropping the communication would know p, g, A and B, and would have to compute a and b from them, that is, solve the equations $g^a \bmod p = A$ and $g^b \bmod p = B$. This is known as the *discrete logarithm problem* and no known technique allows one to solve it effectively if p is large enough and "well chosen". Note that the fact that g is a primitive root of p maximises the number of integers of the form $g^i \bmod p$, hence maximises the number of possible solutions to those equations.

3 The RSA (Rivest-Shamir-Adleman) protocol

This protocol is asymmetric. It lets anyone encrypt a message that only Alice can decrypt.

Alices chooses two secret large prime numbers p and q, as well as a prime number e that is coprime with $(p-1)(q-1)$. (It suffices to let e be a prime number smaller than $(p-1)(q-1)$. In practice, 65537, the largest known prime number of the form $2^{2^n}+1$, is often chosen.) Set $N = pq$. So $\phi(N) = (p-1)(q-1)$. Alices computes the inverse d of e modulo $\phi(N)$, that she keeps secret together with p and q. She publishes N and e, the *public key*. Bob encodes a message that takes the form of a positive integer m that is coprime to N (since p and q are large, this is guaranteed if m is small) as $c = m^e \bmod N$. Alice can then decode c by computing $c^d \bmod N$. Indeed, since $de = 1 \bmod \phi(N)$, $de-1$ is of the form $k\phi(N)$ for some integer k. Using Euler's theorem, it follows that:

$$\begin{aligned} c^d \bmod N = (m^e)^d \bmod N = m^{ed} \bmod N = mm^{ed-1} \bmod N = mm^{k\phi(N)} \bmod N \\ = m(m^{\phi(N)})^k \bmod N = m1^k \bmod N = m. \end{aligned}$$

To compute m, it seems necessary to be able to compute $c^d \bmod N$, hence know d, which seems to require knowing $\phi(N)$, which seems to require knowing $p-1$ and $q-1$, which seems to require knowing p and q. But there is no known method to effectively *factorise* N when the factors p and q of N are large enough and "well chosen".

4 Implementation

Let us write a function that illustrates the Diffie Hellman protocol, where the secret numbers a and b chosen by Alice and Bob, respectively, are between 0 and 99 and randomly generated:

[2]:
```
def diffie_hellman(p, g):
    print(f'Alice lets Bob know that p = {p} and g = {g}.')
    a = randrange(100)
    A = g ** a % p
    b = randrange(100)
    B = g ** b % p
    print(f'Alice sends {A} to Bob.')
    print(f'Bob sends {B} to Alice.')
    print(f'Alice computes the secret code as {B ** a % p}.')
    print(f'Bob computes the secret code as {A ** b % p}.')
```

`diffie_hellman()` has to be provided as arguments a prime number and a primitive root of that prime number. Let us verify that 3 is a primitive root of the prime number 31 (we have not provided any method to compute primitive roots):

[3]:
```
len(set(3 ** i for i in range(31)))
```

[3]:
```
31
```

With this choice of prime number and primitive root, let us get 10 times confirmation that Alice and Bob compute the same secret code:

[4]:
```
for _ in range(10):
    diffie_hellman(31, 3)
    print()
```

```
Alice lets Bob know that p = 31 and g = 3.
Alice sends 17 to Bob.
Bob sends 13 to Alice.
Alice computes the secret code as 22.
Bob computes the secret code as 22.

Alice lets Bob know that p = 31 and g = 3.
Alice sends 21 to Bob.
Bob sends 12 to Alice.
Alice computes the secret code as 13.
Bob computes the secret code as 13.

Alice lets Bob know that p = 31 and g = 3.
Alice sends 17 to Bob.
Bob sends 17 to Alice.
Alice computes the secret code as 12.
Bob computes the secret code as 12.

Alice lets Bob know that p = 31 and g = 3.
Alice sends 5 to Bob.
```

```
Bob sends 4 to Alice.
Alice computes the secret code as 1.
Bob computes the secret code as 1.

Alice lets Bob know that p = 31 and g = 3.
Alice sends 6 to Bob.
Bob sends 22 to Alice.
Alice computes the secret code as 26.
Bob computes the secret code as 26.

Alice lets Bob know that p = 31 and g = 3.
Alice sends 12 to Bob.
Bob sends 24 to Alice.
Alice computes the secret code as 17.
Bob computes the secret code as 17.

Alice lets Bob know that p = 31 and g = 3.
Alice sends 25 to Bob.
Bob sends 2 to Alice.
Alice computes the secret code as 1.
Bob computes the secret code as 1.

Alice lets Bob know that p = 31 and g = 3.
Alice sends 3 to Bob.
Bob sends 12 to Alice.
Alice computes the secret code as 12.
Bob computes the secret code as 12.

Alice lets Bob know that p = 31 and g = 3.
Alice sends 12 to Bob.
Bob sends 19 to Alice.
Alice computes the secret code as 28.
Bob computes the secret code as 28.

Alice lets Bob know that p = 31 and g = 3.
Alice sends 1 to Bob.
Bob sends 3 to Alice.
Alice computes the secret code as 1.
Bob computes the secret code as 1.
```

Let us now write code to illustrate the RSA protocol. Alice has to compute the inverse d of an integer e modulo an integer k (of the form $\phi(pq)$) with e being coprime to k. So $de \bmod k = 1$, which means that there exists an integer x such that $kx + ed = 1$. We have seen that the extended Euclidian algorithm allows one to compute x and d from k and e. (Here, we are only interested in the value of d):

[5]:
```
def bezout_coefficients(a, b):
    '''
    Returns a pair (x, y) with ax + by = gcd(a, b)
    '''
    if b == 0:
        return 1, 0
    x, y = bezout_coefficients(b, a % b)
    return y, x - (a // b) * y
```

[6]:
```
# 123987659 and 98765434289 are prime,
# hence Alice can choose them for p and q, respectively.
k = (123987659 - 1) * (98765434289 - 1)
e = 65537
x, d = bezout_coefficients(k, e)
x, d
k * x + e * d
```

[6]:
```
(20565, -3842603649641703007)
```

[6]:
```
1
```

To decode the coded message c that Bob sent her, Alice has to compute $c^d \bmod N$. When d is very large, this can be done effectively thanks to the equalities:

$$c^{2i} \bmod N = (c^2 \bmod N)^i \quad \text{and} \quad c^{2i+1} \bmod p = ((c^2 \bmod N)^i \times c) \bmod N.$$

[7]:
```
def modular_exponentiation(x, n, p):
    '''
    Returns x^n (mod p)
    '''
    if n == 0:
        return 1
    y = modular_exponentiation((x * x) % p, n // 2, p)
    if n % 2:
        y = (y * x) % p
    return y
```

With `bezout_coefficients()` and `modular_exponentiation()` in hand, it is easy to write a function that illustrates the RSA protocol, where the message that Bob wishes to send to Alice after it has been encoded is a randomly generated integer between 0 and 99:

[8]:
```
def RSA(p, q):
    N = p * q
    phi_of_N = (p - 1) * (q - 1)
    e = 65537
    print(f'Alice publishes public key: N = {N}, e = {e}')
    _, d = bezout_coefficients(phi_of_N, e)
    # d does not have to be smaller than phi_of_N, but it has to be
    # positive.
    d %= phi_of_N
    m = randrange(100)
```

```
    c = m ** e % N
    print(f'Bob encodes {m} as {c}.')
    print(f'Alice decodes {c} as {modular_exponentiation(c, d, N)}.')
```

Choosing 123987659 for p and 98765434289 for q, let us get 10 times confirmation that Alice corrects decodes the encoded message that Bob addresses her:

[9]:
```
for _ in range(10):
    RSA(123987659, 98765434289)
    print()
```

```
Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 62 as 7932971427662815301.
Alice decodes 7932971427662815301 as 62.

Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 29 as 10750341869177849829.
Alice decodes 10750341869177849829 as 29.

Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 57 as 1582771826357032778.
Alice decodes 1582771826357032778 as 57.

Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 69 as 9935791571082425513.
Alice decodes 9935791571082425513 as 69.

Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 24 as 11557713276650351962.
Alice decodes 11557713276650351962 as 24.

Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 40 as 1614359155563093754.
Alice decodes 1614359155563093754 as 40.

Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 74 as 11289891931434069946.
Alice decodes 11289891931434069946 as 74.

Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 79 as 2436093020598851790.
Alice decodes 2436093020598851790 as 79.

Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 60 as 3003536424918532500.
Alice decodes 3003536424918532500 as 60.

Alice publishes public key: N = 12245694987611439451, e = 65537
Bob encodes 37 as 3842600624329825472.
```

Alice decodes 3842600624329825472 as 37.