Computer Science COMP-547A
Cryptography and Data Security

Claude Crépeau

These notes are, largely, transcriptions by Anton Stiglic of class notes from the former course *Cryptography and Data Security (308-647A)* that was given by prof. Claude Crépeau at McGill University during the autumns of 1998 and 1999. These notes are updated and revised each year by Prof. Claude Crépeau.

Latest update September 5, 2008.

1 Basic Number Theory

1.1 Definitions

Divisibility:

$$a|b \iff \exists k \in Z\ [b = ak]$$

Congruences:

$$a \equiv b \pmod{n} \iff n|(a-b)$$

Modulo operator: (Maple `irem, mod`)

$$b \bmod n = \min\{a \geq 0 : a \equiv b \pmod{n}\}$$

Division operator: (Maple `iquo`)

$$b \text{ div } n = \lfloor b/n \rfloor = \frac{b - (b \bmod n)}{n}$$

Greatest Common Divider: (Maple `igcd, igcdex`)

$$g = gcd(a,b) \iff g|a, g|b \text{ and } [g'|a, g'|b \Rightarrow g'|g]$$

Euler's Phi function: (Maple `phi`)

$$\phi(n) = \#\{a : 0 < a < n \text{ and } gcd(a,n) = 1\}$$

Note. $\phi(p) = p - 1, \phi(pq) = (p-1)(q-1)$, where p and q are primes. If $n = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$ then $\phi(n) = (p_1 - 1)p_1^{e_1-1}(p_2 - 1)p_2^{e_2-1} \ldots (p_k - 1)p_k^{e_k-1}$.

1.2 Efficient basic operations

For the basic operations of $+, -, \times, \bmod, \text{div}$ one may use standard "high school" algorithms reducing the work load by the following rules:

$$a \left\{ \begin{array}{c} + \\ - \\ \times \end{array} \right\} b \bmod n = \left((a \bmod n) \left\{ \begin{array}{c} + \\ - \\ \times \end{array} \right\} (b \bmod n) \right) \bmod n$$

The standard "high school" algorithms are precisely described in Knuth (Vol 2). For very large numbers, special purpose divide-and-conquer algorithms may be used for better efficiency of $\times, \bmod, \text{div}$. Consult the algorithmics book of Brassard-Bratley for these.

1.3 Fast modular exponentiation

The idea behind this algorithm is to maintain in each iteration the value of the expression $xa^e \bmod n$ while reducing the exponent e by a factor 2.

Algorithm 1.1 ($a^e \bmod n$)

1: $x \leftarrow 1$,

2: **WHILE** $e > 0$ **DO**

3: **IF** e *is odd* **THEN** $x \leftarrow ax \bmod n$,

4: $a \leftarrow a^2 \bmod n$, $e \leftarrow e$ *div* 2,

5: **ENDWHILE**

6: **RETURN** x.

(Maple `x&^e mod n`)

1.4 GCD calculations and multiplicative inverses

Note. $gcd(a,b) = g \rightarrow \exists_{x,y} \in Z$ such that $g = ax + by$. The following recursive definition is based on the property $gcd(a,b) = gcd(a, b-a)$.

$$gcd(a,b) = \begin{cases} a & \text{if } b = 0 \\ gcd(b, a \bmod b) & \text{otherwise} \end{cases}$$

The idea behind the following iterative algorithm is to maintain in each iteration the relations $g = ax + by$ and $g' = ax' + by'$ while reducing the value of g. At the end of the algorithm, the value of g is $gcd(a,b)$. The final value of x is such that $ax \equiv g \pmod{b}$. When $gcd(a,b) = 1$, we find that x is the multiplicative inverse of a modulo b.

Algorithm 1.2 (Euclide $gcd(a,b)$)

$g \leftarrow a,\ g' \leftarrow b,\ x \leftarrow 1,\ y \leftarrow 0,\ x' \leftarrow 0,\ y' \leftarrow 1,$

WHILE $g' > 0$ **DO**

$k \leftarrow g\ div\ g',$

$(\hat{g}, \hat{x}, \hat{y}) \leftarrow (g, x, y) - k(g', x', y'),$

$(g, x, y) \leftarrow (g', x', y'),$

$(g', x', y') \leftarrow (\hat{g}, \hat{x}, \hat{y}),$

ENDWHILE

RETURN (g, x, y).

(Maple `igcd, igcdex, x^(-1) mod n, 1/x mod n`)

1.5 Solving linear congruentials

A linear congruential is an expression of the form

$$c \equiv ax + b \pmod{n}$$

for known a, b, c, n and unknown variable x. Clearly, we can solve for x whenever $gdc(a, n) = 1$ since in that case $a^{-1} \pmod{n}$ exists and thus

$$x \equiv (c - b)\ a^{-1} \pmod{n}.$$

However, when $gdc(a, n) = g > 1$ the situation becomes less trivial. If it is the case that $g|(c - b)$ as well we can solve the following system instead:

$$(a/g)\ x' \equiv (c - b)/g \pmod{n/g}.$$

Since $gdc(a/g, n/g) = 1$, in that case $(a/g)^{-1} \pmod{n/g}$ exists we can solve for x'

$$x' \equiv (c - b)/g\ \ (a/g)^{-1} \pmod{n/g}.$$

Note however that no solution exists if $g \not| (c - b)$.

Finally, we know that a solution x modulo n must satisfy $x \equiv x' \pmod{n/g}$. Thus we can write

$$x = x' + kn/g$$

and consider all such x with $0 \leq k < g$. All these posibilties for x will be valid solutions to the original system.

1.6 Quadratic Residues

Quadratic residues modulo n are the integers with an integer *square root* modulo n (Maple `quadres`):

$$QR_n = \{a : gcd(a,n) = 1, \exists r[a \equiv r^2 \pmod{n}]\}$$
$$QNR_n = \{a : gcd(a,n) = 1, \forall r[a \not\equiv r^2 \pmod{n}]\}$$

Example:

$$QR_{17} = \{1, 2, 4, 8, 9, 13, 15, 16\}$$
$$QNR_{17} = \{3, 5, 6, 7, 10, 11, 12, 14\}$$

since

$$\{1^2, 2^2, 3^2, 4^2, 5^2, 6^2, 7^2, 8^2, 9^2, 10^2, 11^2, 12^2, 13^2, 14^2, 15^2, 16^2\} \equiv$$
$$\{1, 2, 4, 8, 9, 13, 15, 16\} \pmod{17}.$$

Theorem 1.1 *Let p be an odd prime number*

$$\#QR_p = \#QNR_p = (p-1)/2.$$

1.7 Legendre and Jacobi Symbols

For an odd prime number p, we define the Legendre symbol (Maple `legendre`) as

$$\left(\frac{a}{p}\right) = \begin{cases} +1 & \text{if } a \in QR_p \\ -1 & \text{if } a \in QNR_p \\ 0 & \text{if } p|a \end{cases}$$

For any integer $n = p_1 p_2 ... p_k$, we define the Jacobi symbol (Maple `jacobi`) (a generalization of the Legendre symbol) as

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)...\left(\frac{a}{p_k}\right)$$

Properties

$$\left(\frac{1}{n}\right) = +1$$

$$\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$$

$$\left(\frac{a}{n}\right) = \left(\frac{a \bmod n}{n}\right)$$

For n odd

$$\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$$

$$\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}$$

For a, n odd and such that $gcd(a, n) = 1$

$$\left(\frac{a}{n}\right)\left(\frac{n}{a}\right) = (-1)^{(n-1)(a-1)/4}$$

Algorithm 1.3 **(** $Jacobi(a, n)$ **)**

```
1:if a ≤ 1 then return a
      else  if a is odd then if a ≡ n ≡ 3   (mod 4)
                               then return −Jacobi(n mod a, a)
                               else  return +Jacobi(n mod a, a)
                        else   if n ≡ ±1   (mod 8)
                               then return +Jacobi(a/2, n)
                               else  return −Jacobi(a/2, n)
```

This algorithm runs in $O((lg\ n)^2)$ bit operations.

1.8 Fermat-Euler

Theorem 1.2 (Fermat) *Let p be a prime number and a be an integer not a multiple of p, then*

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

Theorem 1.3 *Let p be a prime number and a be an integer, then*

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \pmod{p}.$$

Theorem 1.4 (Euler) *Let n be an integer and a another integer such that $gcd(a, n) = 1$, then*

$$a^{\phi(n)} \equiv 1 \pmod{n}.$$

1.9 Extracting Square Roots modulo p

Theorem 1.5 *For prime numbers $p \equiv 3 \pmod 4$ and $a \in QR_p$, we have that $r = a^{(p+1)/4} \bmod p$ is a square root of a.*

Proof.

$$\begin{aligned}(a^{(p+1)/4)})^2 &\equiv a^{(p-1)/2} \cdot a \pmod{p} \\ &\equiv a \pmod{p} (\text{Fermat, sec. 1.2})\end{aligned}$$

For prime numbers $p \equiv 1 \pmod 4$ and $a \in QR_p$, there (only) exists an efficient *probabilistic* algorithm. We present one found in the algorithmics book of Brassard-Bratley:

Algorithm 1.4 (rootLV(x, p, VAR y, VAR success))

1: $a \leftarrow uniform(1 \ldots p-1)$

2: **IF** $a^2 \equiv x \; mod \; p$ {*very unlikely*}

3: **THEN** *success* $\leftarrow$ *true*, $y \leftarrow a$

4: **ELSE** *compute c and d such that* $0 \leq c \leq p-1, 0 \leq d \leq p-1$, *and* $(a + \sqrt{x})^{(p-1)/2} \equiv c + d\sqrt{x} \; mod \; p$

5: **IF** $d = 0$ **THEN** *success* $\leftarrow$ *false*

6: **ELSE** $c = 0$, *success* $\leftarrow$ *true*,

7: *compute y such that* $1 \leq y \leq p-1$ *and* $d \cdot y \equiv 1 \bmod p$

1.10 Chinese Remainder Theorem

Theorem 1.6 (Chinese Remainder (Maple `chrem`)) *Let $m_1, m_2, ..., m_r$ be r positive integers such that $gcd(m_i, m_j) = 1$ for $1 \leq i < j \leq r$ and let*

$a_1, a_2, ..., a_r$ be integers. The system of r congruences $x \equiv a_i \pmod{m_i}$, for $1 \le i \le r$ has a unique solution modulo $M = m_1 m_2 ... m_r$ which is given by

$$x = \sum_{i=1}^{r} a_i M_i y_i \bmod M$$

where $M_i = M/m_i$ and $y_i = M_i^{-1} \bmod m_i$, for $1 \le i \le r$.

1.11 Application: Extracting Square Roots modulo n

We want to solve $x^2 \equiv a \pmod{n}$ for x knowing $n = pq$. We first solve modulo p and q and find solutions to

$$\begin{aligned} {x_p}^2 &\equiv a \pmod{p} \\ {x_q}^2 &\equiv a \pmod{q}. \end{aligned}$$

We then consider the simultaneous congruences

$$\begin{aligned} x \equiv x_p \pmod{p} &\quad \Longleftrightarrow p | x^2 - a \\ x \equiv x_q \pmod{q} &\quad \underbrace{\Longleftrightarrow q | x^2 - a} \\ &\quad \Rightarrow p \cdot q = n | x^2 - a \\ &\quad \Rightarrow x^2 \equiv a \pmod{n} \end{aligned}$$

We can now solve x by the chinese remainder theorem.

Definition 1.7 (SQROOT) *The square root modulo n problem can be stated as follows:*
given a composite integer n and $a \in QR_n$, find a square root of a mod n.

(Maple `msqrt`)

Theorem 1.8 *SQROOT is polynomialy equivalent to FACTORING (see def. section 12.1).*

Proof idea: the above construction shows that if we know the factorization of n, we can extract square roots modulo each prime factor of n and then recombine using the Chinese Remainder Theorem.

If we can extract square roots modulo n, we can split n in two factors $n = uv$ by repeating the following algorithm: Pick a random integer a and extract the square root of $a^2 \bmod n$, say a'. If $a' \equiv \pm a \pmod{n}$ then try again, else set $u = gcd(a + a', n)$ and $v = gcd(a - a', n)$. The probability of the second case is at least 1/2.

1.12 ***Extracting Square Roots modulo p^e

If we have a solution r to $r^2 \equiv x \pmod{p}$, how do we find a solution s to $s^2 \equiv x \pmod{p^e}$ for $e > 1$?

The chinese remainder theorem does not apply here. We have to figure things out in a different way.

First, consider the case $e = 2$. Since $r^2 \equiv x \pmod{p}$, there exists an integer $m = (r^2 - x)/p$ such that $r^2 - x = mp$. Suppose the solution mod p^2 is of the form $s = r + kp$ for some integer k. Let's expand s^2 :

$$s^2 = (r + kp)^2 = r^2 + 2rkp + (kp)^2 = mp + x + 2rkp + (kp)^2$$

and therefore

$$s^2 \equiv x + (m + 2rk) * p \pmod{p^2}.$$

We find a solution s by making $m + 2rk$ a multiple of p so that

$$(m + 2rk) * p \equiv 0 \pmod{p^2}.$$

The following value of k will acheive our goal

$$k \equiv -m * (2r)^{-1} \pmod{p}$$

and thus remembering $s = r + kp$ we get

$$s = r - (m * (2r)^{-1} \bmod p) * p$$

and finally remembering $m = (r^2 - x)/p$ we obtain a solution

$$s = r + (x - r^2) * ((2r)^{-1} \bmod p).$$

Second, notice that the same exact reasoning allows to go from the case p^e to the case p^{2e}, meaning that any solution r to $r^2 \equiv x \pmod{p^e}$, can be transformed to a solution $s = r + kp^e$ of $s^2 \equiv x \pmod{p^{2e}}$.

Using this argument i times allows to start from a solution r to $r^2 \equiv x \pmod{p}$, and find a solution s to $s^2 \equiv x \pmod{p^{2^i}}$.

Finally, to solve the general problem where e is not necessarily a power of 2, let i be the smallest integer such that $2^i \geq e$. From a solution r to $r^2 \equiv x \pmod{p}$, find a solution to $s^2 \equiv x \pmod{p^{2^i}}$ and since $p^e | p^{2^i}$ this same solution s will also work $\bmod p^e$.

1.13 Prime numbers

If we want a random prime (Maple `rand, isprime`) of a given size, we use the following theorem to estimate the number of integers we must try before finding a prime. Let $\pi(n) = \#\{a : 0 < a \leq n \text{ and } a \text{ is prime}\}$.

Theorem 1.9 $\displaystyle\lim_{n\to\infty} \frac{\pi(n)\log n}{n} = 1$

To decide whether a number n is prime or not we rely on Miller-Rabin's probabilistic algorithm. This algorithm introduces the notion of "pseudo-primality" base a. Miller defined this test as an extension of Fermat's test. If the Extended Riemann Hypothesis is true than it is sufficient to use the test with small values of a to decide whether a number n is prime or composite. However the ERH is not proven and we use the test in a probabilistic fashion as suggested by Rabin.

Algorithm 1.5 ($Pseudo(a, n)$)

IF $gcd(a, n) \neq 1$ **THEN RETURN** *"composite"*,

Let t be an odd number and s a positive integer such that $n - 1 = t2^s$

$x \leftarrow a^t \bmod n$, $y \leftarrow n - 1$,

FOR $i \leftarrow 0$ **TO** s

IF $x = 1$ **AND** $y = n - 1$ **THEN RETURN** *"pseudo"*,

$y \leftarrow x$, $x \leftarrow x^2 \bmod n$,

ENDFOR

RETURN *"composite"*.

It is easy to show that if n is prime, then $Pseudo(a, n)$ returns "pseudo" for all a, $0 < a < n$. Rabin showed that if n is composite, then $pseudo(a, n)$ returns "composite" for at least $3n/4$ of the values of a, $0 < a < n$.

Theorem 1.10

$$\#\{a : Pseudo(a, n) = \text{"pseudo"}\} \begin{cases} = \phi(n) & = n - 1 & \textit{if } n \textit{ is prime} \\ \leq \phi(n)/4 & \leq (n-1)/4 & \textit{if } n \textit{ is composite.} \end{cases}$$

To increase the certainty we may repeat the above algorithm as follows.

Algorithm 1.6 (Miller-Rabin $prime(n,k)$ **)**

1: FOR $i \leftarrow 1$ **TO** k

2: *Pick a random element a,* $0 < a < n$*,*

3: IF $pseudo(a,n) =$ *"composite"* **THEN RETURN** *"composite",*

4: ENDFOR

5: RETURN *"prime".*

We easily deduce that if n is prime, then $prime(n,k)$ always returns "prime" and that if n is composite, then $prime(n,k)$ returns "composite" with probability at least $1-(1/4)^k$. Thus when the algorithm *prime* returns "composite", it is always a correct verdict. However when it returns "prime" it remains a very small probability that this verdict is wrong.

In August of 2002, Agrawal, Kayal, and Saxena, announced the discovery of a *deterministic* primality test running in polynomial time. Unfortunately this test is too slow in practice... its running time being $O(|n|^{12})$.

1.14 Quadratic Residuosity problem

Definition 1.11

$$J_n := \{a \in \mathbb{Z}_n \mid \left(\frac{a}{n}\right) = 1\}$$

Theorem 1.12 *Let n be a product of two distinct odd primes p and q. Then we have that* $a \in QR_n$ *iff* $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = 1$.

Definition 1.13 *The quadratic residuosity problem (QRP) is the following: given an odd composite integer n and* $a \in J_n$*, decide whether or not a is a quadratic residue modulo n.*

Definition 1.14 *(pseudosquare) Let* $n \geq 3$ *be an odd integer. An integer a is said to be a pseudosquare modulo n if* $a \in QNR_n \bigcap J_n$.

Remark: If n is a prime, then it is easy to decide if a is in QR_n, since $a \in QR_n$ iff $a \in J_n$, and the Legendre symbol can be efficiently computed by

algorithm 1.3.
If n is a product of two distinct odd primes p and q, then it follows from theorem 1.12 that if $a \in J_n$, then $a \in QR_n$ iff $\left(\frac{a}{p}\right) = 1$.

If we can factor n, then we can find out if $a \in QR_n$ by computing the Legendre symbol $\left(\frac{a}{p}\right)$.
If the factorization of n is unknown, then there is no efficient algorithm known to decide if $a \in QR_n$.
This leads to the Goldwasser-Micali probabilistic encryption algorithm:
Init: *Alice* starts by selecting two large distinct prime numbers p and q. She then computes $n = pq$ and selects a pseudosquare y. n and y will be public, p and q private.

Algorithm 1.7 (Goldwasser-Micali probabilistic encryption)

1: *Represent message m in binary ($m = m_1 m_2 \dots m_t$).*
2: **FOR** $i = 1$ **TO** t **DO**
3: **Pick** $x \in_R \mathbb{Z}_n^*$
4: $c_i \leftarrow y^{m_i} x^2 \; mod \; n$
5: **RETURN** $c = c_1 c_2 \dots c_t$

Algorithm 1.8 (Goldwasser-Micali decryption)

1: **FOR** $i = 1$ **TO** t **DO**
2: $e_i \leftarrow \left(\frac{c_i}{p}\right)$ *using algo 1.3.*
3: **IF** $e_i = 1$ **THEN** $m_i \leftarrow 0$ **ELSE** $m_i \leftarrow 1$
4: **RETURN** $m = m_1 m_2 \dots m_t$

2 Finite Fields

2.1 Prime Fields

Let p be a prime number. The integers $0, 1, 2, ..., p - 1$ with operations $+ \bmod p$ et $\times \bmod p$ constitute a field $\mathcal{F}_p$ of p elements.

- contains an additive neutral element (0)
- each element e has an additive inverse $-e$
- contains an multiplicative neutral element (1)
- each non-zero element e has a multiplicative inverse e^{-1}
- associativity
- commutativity
- distributivity

Examples $\mathcal{F}_2 = (\{0, 1\}, \oplus, \wedge)$. $\mathcal{F}_5 = (\{0, 1, 2, 3, 4\}, +, \times)$ defined by

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Other kind of finite fields for numbers q not necessarily prime exist (Maple `GF`). This is studied in another section. In general we refer to $\mathcal{F}_q$ for a finite field, but you may think of the special case $\mathcal{F}_p$ if you do not wish to find out about the general field construction.

2.2 Primitive Elements

In all finite fields $\mathcal{F}_q$ (and some groups in general) there exists a *primitive element*, that is an element g of the field such that $g^1, g^2, ..., g^{q-1}$ enumerate all of the $q-1$ non-zero elements of the field. We use the following theorem to find a primitive element over $\mathcal{F}_q$.

Theorem 2.1 *Let $l_1, l_2, ..., l_k$ be the prime factors of $q-1$ and $m_i = (q-1)/l_i$ for $1 \le i \le k$. An element g is* primitive *over $\mathcal{F}_q$ if and only if*

- $g^{q-1} = 1$
- $g^{m_i} \ne 1$ *for* $1 \le i \le k$

Algorithm 2.1 (*Primitive*(q))

1: *Let $l_1, l_2, ..., l_k$ be the prime factors of $q-1$ and $m_i = \frac{q-1}{l_i}$ for $1 \le i \le k$,*

2: **REPEAT**

3: *pick a random non-zero element g of $\mathcal{F}_q$,*

4: **UNTIL** $g^{m_i} \ne 1$ *for* $1 \le i \le k$,

5: **RETURN** g.

(Maple `primroot, G[PrimitiveElement]`)

We use the following theorems to estimate the number of field elements we must try in order to find a random primitive element.

Theorem 2.2 $\#\{g : g$ *is a primitive element of* $\mathcal{F}_q\} = \phi(q-1)$.

Theorem 2.3 $\liminf_{n\to\infty} \dfrac{\phi(n)\log\log n}{n} = e^{-\gamma} \approx 0.5614594836$

Example: 2 is a primitive element of $\mathcal{F}_5$ since $\{2, 2^2, 2^3, 2^4\} = \{2, 4, 3, 1\}$.

Relation to Quadratic residues As an interesting note, if g is a primitive element of the field $\mathcal{F}_p$, for a prime p, then we have:

$$QR_p = \{g^{2i} \bmod p : 0 \leq i < (p-1)/2\}$$
$$QNR_p = \{g^{2i+1} \bmod p : 0 \leq i < (p-1)/2\}$$

in other words, the quadratic residues are the even powers of g while the quadratic non-residues are the odd powers of g.

Factoring $q-1$... In general, it may be difficult to factor $q-1$. It will therefore be only possible to find a primitive element for fields $\mathcal{F}_q$ for which the factorization of $q-1$ is known. However, if we are after a large field with a random number of elements Eric Bach has devised an efficient probabilistic algorithm to generate random integers of a given size with known factorization. Recently, Adam Kalai has invented a somewhat slower algorithm that is much simpler. Suppose we randomly select r with its factorization using Bach's or Kalai's algorithm. We may check whether $r+1$ is a prime or a prime power. In this case a finite field of $r+1$ elements is obtained and a primitive element may be computed.

Algorithm 2.2 (Kalai $randfact(n)$ **)**

1: *Generate a sequence* $n \geq s_1 \geq s_2 \geq ... \geq s_\ell = 1$ *by picking* $s_1 \in_R \{1, 2, ..., n\}$ *and* $s_{i+1} \in_R \{1, 2, ..., s_i\}$, *until reaching 1.*

2: *Let* r *be the product of the prime* s_i*'s.*

3: IF $r \leq n$ **THEN** *with probability* r/n **RETURN** $(r, \{prime\ s_i\text{'s}\})$.

4: *Otherwise,* **RESTART**.

Theorem 2.4 *Let* $M_n = \prod_{p \leq n}(1 - 1/p)$.
The probability of producing r *at step 2 is* M_n/r.

Thus by outputting r with probability r/n in step 3, each possible value is generated with equal probability $\frac{M_n}{r}\frac{r}{n} = \frac{M_n}{n}$. The overall probability that some small enough r is produced and chosen in step 3 is $\sum_{1 \leq r \leq n} \frac{M_n}{n} = M_n$.

Theorem 2.5 $\lim_{n \to \infty} M_n \log n = e^{-\gamma} \approx 0.5614594836$

2.3 Polynomials over a field

A polynomial over $\mathcal{F}_p$ is specified by a finite sequence $(a_n, a_{n-1}, ..., a_1, a_0)$ of elements from $\mathcal{F}_p$, with $a_n \neq 0$. The number n is the degree of the polynomial. We have operations $+, -, \times$ on polynomials analogous to the similar integer operations. Addition and subtraction are performed componentwise using the addition $+$ and subtraction $-$ of the field $\mathcal{F}_p$.

Products are computed by adding all the products of coefficients associated to pairs of exponents adding to a specific exponent.

Example:

$$
\begin{aligned}
&(x^4 + x + 1) \times (x^3 + x^2 + x) \\
&\quad = \; x^4 \times (x^3 + x^2 + x) + x \times (x^3 + x^2 + x) + 1 \times (x^3 + x^2 + x) \\
&\quad = \; (x^7 + x^6 + x^5) + (x^4 + x^3 + x^2) + (x^3 + x^2 + x) \\
&\quad = \; x^7 + x^6 + x^5 + x^4 + (1+1)x^3 + (1+1)x^2 + x \\
&\quad = \; x^7 + x^6 + x^5 + x^4 + x
\end{aligned}
$$

We also have operations $g(x)$ mod $h(x)$ (Maple `modpol, quo`) and $g(x)$ div $h(x)$ (Maple `rem`) defined as the polynomials $r(x)$ and $q(x)$ such that $g(x) = q(x)h(x) + r(x)$ with $deg(r) < deg(h)$. They are obtained by formal division of $g(x)$ by $h(x)$ similar to what we do with integers.

Example:

$$
\begin{aligned}
x^7 + x^6 + x^5 + x^4 + x &= (x^2) \times (x^5 + x^2 + 1) + (x^6 + x^5 + x^2 + x) \\
&= (x^2 + x) \times (x^5 + x^2 + 1) + (x^5 + x^3 + x^2) \\
&= (x^2 + x + 1) \times (x^5 + x^2 + 1) + (x^3 + 1)
\end{aligned}
$$

thus

$$
(x^7 + x^6 + x^5 + x^4 + x) \text{ mod } (x^5 + x^2 + 1) = x^3 + 1
$$
$$
(x^7 + x^6 + x^5 + x^4 + x) \text{ div } (x^5 + x^2 + 1) = x^2 + x + 1
$$

Exponentiations for integer powers modulo a polynomial are computed using an analogue of algorithm 1.1 (Maple `powermod`) and *gcd* (Maple `gcd`) of polynomials or multiplicative inverses (Maple `gcdex, modpol(1/x,q(x),x,p)`) are computed using an analogue of algorithm 1.2.

$\mathcal{F}_2$	
$x+1$	x^9+x^4+1
x^2+x+1	$x^{10}+x^3+1$
x^3+x+1	$x^{11}+x^2+1$
x^4+x+1	$x^{12}+x^6+x^4+x+1$
x^5+x^2+1	$x^{13}+x^4+x^3+x+1$
x^6+x+1	$x^{14}+x^{10}+x^6+x+1$
x^7+x^3+1	$x^{15}+x+1$
$x^8+x^4+x^3+x^2+1$	$x^{16}+x^{12}+x^3+x+1$

Figure 1: Irreducible polynomials over $\mathcal{F}_2$.

$\mathcal{F}_3$	$\mathcal{F}_5$	$\mathcal{F}_7$
$x+1$	$x+1$	$x+1$
x^2+x+2	x^2+x+2	x^2+x+3
x^3+2x+1	x^3+3x+2	x^3+3x+2
x^4+x+2	x^4+x^2+x+2	
x^5+2x+1		
x^6+x+2		

Figure 2: Irreducible polynomials over $\mathcal{F}_3$,$\mathcal{F}_5$,$\mathcal{F}_7$.

2.4 Irreducible Polynomials

A polynomial $g(x)$ is *irreducible* (Maple `irreduc`) if it is not the product of two polynomials $h(x), k(x)$ of lower degrees. We use the following theorem to find irreducible polynomials.

Theorem 2.6 *Let $l_1, l_2, ..., l_k$ be the prime factors of n and $m_i = n/l_i$ for $1 \leq i \leq k$. A polynomial $g(x)$ of degree n is* irreducible *over $\mathcal{F}_p$ iff*

- $g(x) | x^{p^n} - x$
- $gcd(g(x), x^{p^{m_i}} - x) = 1$ *for* $1 \leq i \leq k$

Algorithm 2.3 (Rabin $Irr(p, n)$)

1: *let* $l_1, l_2, ..., l_k$ *be the prime factors of* n *and* $m_i = n/l_i$ *for* $1 \leq i \leq k$*,*

2: REPEAT

3: *pick a random polynomial* $h(x)$ *of degree* $n-1$ *over* $\mathcal{F}_p$*,*
$g(x) \leftarrow x^n + h(x)$,

4: UNTIL $x^{p^n} \bmod g(x) = x$ *and* $gcd(g(x), x^{p^{m_i}} - x) = 1$ *for* $1 \leq i \leq k$*,*

5: RETURN g.

We use the following theorem to estimate the number of polynomials we have to try on average before finding one that is irreducible.

Theorem 2.7 *Let* $m(n)$ *be the number of irreducible polynomials* $g(x)$ *of degree* n *of the form* $g(x) = x^n + h(x)$ *where* $h(x)$ *is of degree* $n-1$*. We have*

$$\frac{p^n}{2n} \leq \frac{p^n - p^{n/2} \log n}{n} \leq m(n) \leq \frac{p^n}{n}.$$

2.5 General Fields

Let p be a prime number and n a positive integer. We construct a field with p^n elements (Maple `GF`) from the basis field $\mathcal{F}_p$ with p elements.

- The elements of $\mathcal{F}_{p^n}$ are of the form $a_1 a_2 ... a_n$ where a_i is an element of $\mathcal{F}_p$.
- The sum of two elements of $\mathcal{F}_{p^n}$ is defined by

  $$a_1 a_2 ... a_n + b_1 b_2 ... b_n = c_1 c_2 ... c_n$$

  such that $c_i = a_i + b_i$ for $1 \leq i \leq n$.
- The product of two elements of $\mathcal{F}_{p^n}$ is defined by

  $$a_1 a_2 ... a_n \times b_1 b_2 ... b_n = c_1 c_2 ... c_n$$

such that

$$(c_1x^{n-1} + c_2x^{n-2} + ... + c_n) =$$
$$(a_1x^{n-1} + a_2x^{n-2} + ... + a_n) \times (b_1x^{n-1} + b_2x^{n-2} + ... + b_n) \bmod r(x)$$

where $r(x)$ is an irreducible polynomial of degree n over $\mathcal{F}_p$.

Examples computations over $\mathcal{F}_{2^5}$
$10011 + 01110 = (1+0)(0+1)(0+1)(1+1)(1+0) = 11101$

+	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
001	001	000	011	010	101	100	111	110
010	010	011	000	001	110	111	100	101
011	011	010	001	000	111	110	101	100
100	100	101	110	111	000	001	010	011
101	101	100	111	110	001	000	011	010
110	110	111	100	101	010	011	000	001
111	111	110	101	100	011	010	001	000

$10011 \times 01110 = 01001$ since $(x^4+x+1) \times (x^3+x^2+x) \bmod (x^5+x^2+1) = x^3 + 1$.

×	000	001	010	011	100	101	110	111
000	000	000	000	000	000	000	000	000
001	000	001	010	011	100	101	110	111
010	000	010	100	110	011	001	111	101
011	000	011	110	101	111	100	001	010
100	000	100	011	111	110	010	101	001
101	000	101	001	100	010	111	011	110
110	000	110	111	001	101	011	010	100
111	000	111	101	010	001	110	100	011

Figure 3: operations of $\mathcal{F}_{2^3}$

2.6 Application of finite fields: Secret Sharing

A polynomial over $\mathcal{F}_q$ is specified by a finite sequence $(a_n, a_{n-1}, ..., a_1, a_0)$ of elements from $\mathcal{F}_q$, with $a_n \neq 0$. The number n is the degree of the polynomial.

Theorem 2.8 (Lagrange's Interpolation) *Let $x_0, x_1, ..., x_d$ be distinct elements of a field $\mathcal{F}_q$ and $y_0, y_1, ..., y_d$ be any elements of $\mathcal{F}_q$. There exists a unique polynomial $p(x)$ over $\mathcal{F}_q$ with degree $\leq d$ such that $p(x_i) = y_i$ for $1 \leq i \leq n$.*

Algorithm 2.4 ($Interpolation(x_0, x_1, ..., x_d, y_0, y_1, ..., y_d)$)

1: return $\begin{pmatrix} 1 & x_0 & \dots & x_0^d \\ 1 & x_1 & \dots & x_1^d \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_d & \dots & x_d^d \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_d \end{pmatrix}$

Of course the matrix inversion is to be performed over $\mathcal{F}_q$, which means all additions, subtractions and multiplications are calculated within the field, and divisions are performed by multiplying with the multiplicative inverse in the field.

Suppose Alice wants to distribute a secret S among n people $P_1, P_2, \ldots, P_n$ in such a way that any k of them can recover the secret from their joint information, while it remains perfectly secret when any $k-1$ or less of them get together. This is what we call a $[n, k]$-secret sharing scheme.

Algorithm 2.5 (**SSSS**(S))

1: $a_0 \leftarrow S$,

2: **FOR** $i := 1$ **TO** $k-1$ **DO** $a_i \leftarrow uniform(0..p-1)$

3: **FOR** $j := 1$ **TO** n **DO** $s_i \leftarrow a_{k-1}j^{k-1} + \ldots + a_1 j + a_0 \bmod p$

4: **RETURN** $s_1, s_2, \ldots, s_n$.

Let's be a bit more formal. Let S be Alice's secret from the finite set $\{0, 1, 2, \ldots, M\}$ and let p be a prime number greater than M and n, the number of share holders. Shamir's construction of a $[n,k]$-secret sharing scheme is as follows.

Share s_j is given to P_j secretly by Alice. In order to find S, k or more people may construct the matrix from Lagrange's theorem from the distinct values $x_j = j$ and find the unique $(a_0, a_1, \ldots, a_{k-1})$ corresponding to their values $y_j = s_j$.

Theorem 2.9 *For* $0 \leq m \leq n$, *distinct* $j_1, j_2, \ldots, j_m$ *and any* $s_{j_1}, s_{j_2}, \ldots, s_{j_m}$

$$S|[j_1, s_{j_1}], [j_2, s_{j_2}], \ldots, [j_m, s_{j_m}] = \begin{cases} C & \text{if } m \geq k \\ U & \text{if } m < k \end{cases}$$

where C is the constant random variable with $Pr[C = c] = 1$ for one single constant c (meaning that the secret is fully determined), and U is the uniform distribution (meaning that the secret is completely undetermined).

Algorithm 2.6 ($Solve(x_1, x_2, ..., x_m, s_1, s_2, ..., s_m)$)

1:
$$\begin{pmatrix} 1 & x_1 & \ldots & x_1^{k+d} & -s_1 & \ldots & -s_1 x_1^k \\ 1 & x_2 & \ldots & x_2^{k+d} & -s_2 & \ldots & -s_2 x_2^k \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_i & \ldots & x_i^{k+d} & -s_i & \ldots & -s_i x_i^k \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \ldots & x_m^{k+d} & -s_m & \ldots & -s_m x_m^k \end{pmatrix} \begin{pmatrix} n_0 \\ n_1 \\ \vdots \\ n_{k+d} \\ w_0 \\ 1 \\ w_2 \\ \vdots \\ w_k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix}$$