Criptografia Aplicada, 2021/2022

RSA — and related subjects

The Magic Words are Squeamish Ossifrage

Guess who contributed a modest amount of computation time to this collaborative effort.

Security (spoiler)

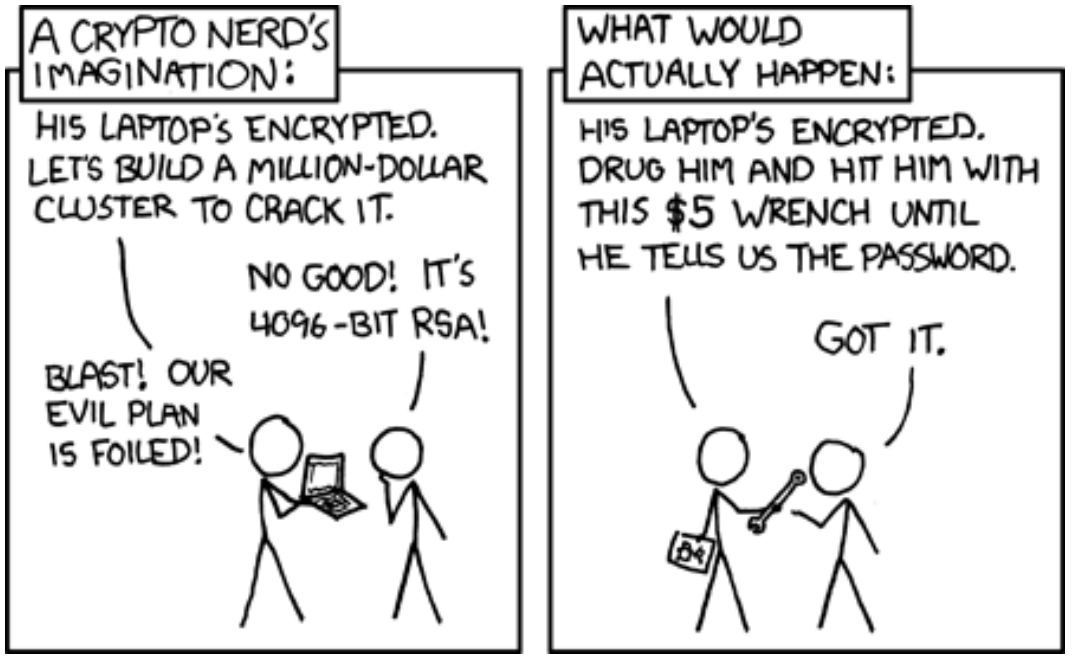
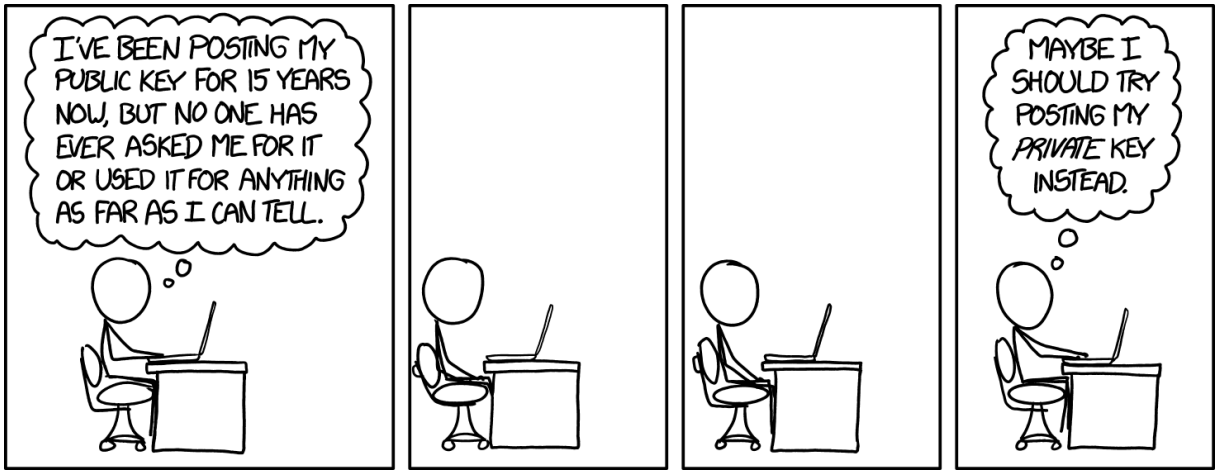


Table of Contents

Goals

- Public-key cryptography
- Sharing secrets
- Doing things without leaking information

Public Key (spoiler)



Means

- Number theory.
- In particular, modular arithmetic. Why? Because:
  - we will be performing computations with a finite set of integers (for example, there is no need to worry about round-off errors);
  - modular arithmetic can be done efficiently in almost all computing devices;
  - and last, but not least, because there exist many number theoretic theorems that have cryptographic applications.

Mathematics is the queen of the sciences and
number theory is the queen of mathematics.
Carl Friedrich Gauss

Programming languages you may use

- `C`, in particular the GNU MP library, also known as `libgmp`
- `C++`, using also the GNU MP library, but with classes and arithmetic operator overloading!
- `Python`
- `Java`, in particular the `BigInteger` class
- `pari-gp` (get it here), because it has everything we will need
- `SageMath` (get it here), because it has everything we will need and its interface uses the Python programming language (but its a big download)

Modular arithmetic

notation	meaning
$m \mid n$	m divides n.
$m \nmid n$	m does not divide n.
$n \equiv r \pmod{m}$	$m \mid (n-r)$, that is, as m divides $n-r$, n and r have the same remainder when divided by m.
$\lfloor x \rfloor$	floor function: largest integer not larger than x.
$n \bmod m$	(binary operator) remainder of n when divided by m (m is called the modulus, which we assume here to be a positive integer). Equal to $n - m\left\lfloor \frac{n}{m} \right\rfloor$. Note that $0 \leqslant r < m$. In C, Python, Java, and pari-gp, it can be computed using the % binary operator (applied to unsigned integers).
$\gcd(a,b)$	greatest common divisor of a and b.
$\operatorname{lcm}(a,b)$	least common multiple of a and b; equal to $ab/\gcd(a,b)$.
$\mathbb{Z}_m$	set of equivalence classes modulo m; slightly abusing the mathematical notation for equivalence classes, $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$

Modular arithmetic examples

- $1 \mid 10$, $5 \mid 20$, $7 \mid 7$, $11 \mid 44$, $3 \nmid 5$
- $17 \equiv 7 \pmod{10}$, $27 \equiv 17 \pmod{10}$, $27 \equiv 7 \pmod{10}$
- $\lfloor 1.1 \rfloor = 1$, $\lfloor 7/3 \rfloor = 2$, $\lfloor -1.1 \rfloor = -2$
- $17 \bmod 6 = 5$, $7 \bmod 6 = 1$, $(17 \times 7) \bmod 6 = (5 \times 1) \bmod 6 = 5$
- $\gcd(15, 25) = 5$, $\gcd(7, 6) = 1$, when n is a positive integer, $\gcd(n, n + 1) = 1$
- $\operatorname{lcm}(15, 25) = 75$, $\operatorname{lcm}(7, 6) = 42$
- modulo m, the set of the integers — $\mathbb{Z}$ — is partitioned into m equivalence classes; we can choose as representative for each equivalence class an integer from the set $\mathbb{Z}_m$; for example, for $m = 5$, we have

  equivalence class with representative 0: $\ldots, -5, \underline{0}, 5, 10, \ldots$
  equivalence class with representative 1: $\ldots, -4, \underline{1}, 6, 11, \ldots$
  equivalence class with representative 2: $\ldots, -3, \underline{2}, 7, 12, \ldots$
  equivalence class with representative 3: $\ldots, -2, \underline{3}, 8, 13, \ldots$
  equivalence class with representative 4: $\ldots, -1, \underline{4}, 9, 14, \ldots$

  The binary $\bmod$ operator we defined in the previous slide computes this representative.

More modular arithmetic examples

Tables for addition (on the left) and multiplication (on the right) modulo 7.

$+$

$a\backslash b$	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

$\times$

$a\backslash b$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

- All elements of $\mathbb{Z}_7$ have a symmetric value; given any a it is also possible to find a b, which is unique, such that $a + b \equiv 0 \pmod{m}$. This is so in general.
- In this case all non-zero elements of $\mathbb{Z}_7$ have inverses. However, this is **not** general. An element a of $\mathbb{Z}_m$ has an inverse if and only if $\gcd(a, m) = 1$. The inverse of a, if it exists, is the (unique in $\mathbb{Z}_m$) b such that $ab \equiv 1 \pmod{m}$.

Modular arithmetic in C

- Addition, for small integers:

```
long add_mod(long a,long b,long m)
{ // assuming that 0 <= a,b < m, return (a+b) mod m
  long r = a + b;
  if(r >= m)
    r -= m;
  return r;
}
```

- Addition, for arbitrary precision integers (using the GNU MP library):

```
#include <gmp.h>
void add_mod(mpz_t r,mpz_t a,mpz_t b,mpz_t m)
{ // assuming that 0 <= a,b < m, compute r = (a+b) mod m
  mpz_add(r,a,b); // r = a+b
  if(mpz_cmp(r,m) >= 0)
    mpz_sub(r,r,m); // r -= m
}
```

Modular arithmetic exercises

Use a program (and perhaps brute force) to compute:

- $(1122334455 \times 6677889900) \bmod 349335433$
- $3^{-1} \bmod 7$ (this one does not require a program but do it anyway, it can be used to check if your program is working properly)
- $4^{-1} \bmod 7$ (neither does this one)
- $3^{-1} \bmod 10$ (neither does this one)
- $271828^{-1} \bmod 314159$ (just to warm up)
- $271828183^{-1} \bmod 314159265$ (now we're cooking!)
- $2718281828459^{-1} \bmod 3141592653590$ (can you handle this one?)
- $271828182845904523536028 75^{-1} \bmod 3141592653589793238462643 4$ (is the teacher sane?)

Fast modular multiplication

A modular multiplication requires a remainder operation, which is a slow operation if the modulus is a general integer. For example, contemporary processors can multiply two 64-bit integers, producing a 128-bit result, with a latency of 3 or 4 clock cycles. But, dividing a 128-bit integer by a 64-bit integer, producing a 64-bit quotient and a 64-bit remainder, is considerably slower (tens of clock cycles).

If the modulus is a power of two, say 2^n, the remainder operation is very fast; the remainder is just the last n bits of the number being remaindered. In 1985, Peter Montgomery came up with a beautiful way to explore this to efficiently perform general remaindering operations without performing an expensive division.

Homework: Read Peter's paper Modular Multiplication Without Trial Division. You can get extra information by searching the internet for "Montgomery modular multiplication".

The greatest common divisor

- Let p_k be the k-th prime number, so that $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, and so on.
- Each positive integer can the factored into prime factors in a unique way (this is the fundamental theorem of arithmetic).
- Let $a = \prod_{k=1}^{\infty} p_k^{a_k}$, where a_k is the number of times p_k divides a. Since a is a finite number, almost all of the a_k values will be zero.
- Likewise of b, let $b = \prod_{k=1}^{\infty} p_k^{b_k}$.
- Then,

$$\gcd(a, b) = \prod_{k=1}^{\infty} p_k^{\min(a_k, b_k)}$$

and

$$\operatorname{lcm}(a, b) = \prod_{k=1}^{\infty} p_k^{\max(a_k, b_k)}$$

- If $\gcd(a, b) = 1$ then a and b are said to be relatively prime (or coprime).

The greatest common divisor (algorithm)

Assume that $a \geqslant 0$ and that $b \geqslant 0$. Then:

- $\gcd(a,b) = \gcd(b,a)$, and so $\gcd(a,b) = \gcd\big(\max(a,b), \min(a,b)\big)$. Thus, by exchanging a with b if necessary, we may assume that $a \geqslant b$.
- as any positive integer divides 0 we have $\gcd(a,0) = a$ for $a > 0$. The mathematicians say that $\gcd(0,0) = 0$, and so we can say that $\gcd(a,0) = a$ as long as $a \geqslant 0$.
- If $a \geqslant b$ then $\gcd(a,b) = \gcd(a-b,b)$. We can keep subtracting b from (the updated) a until it becomes smaller than b, and so $\gcd(a,b) = \gcd(a \bmod b, b)$.

These observations give rise to the following so-called Euclid's algorithm (coded in C, but it can easily be translated to another programming language):

```
long gcd(long a,long b)
{
  while(b != 0) { long c = a % b; a = b; b = c; } return a;
}
```

The GNU MP library has a function, `mpz_gcd`, for this; `pari-gp` does this with the `gcd` function.

The greatest common divisor (example)

Goal: to compute $\mathbf{gcd(273, 715)}$.

- Step 1: $\mathbf{gcd(273, 715) = gcd(715, 273)}$.
- Step 2: $\mathbf{gcd(715, 273) = gcd(715 - 2 \times 273, 273) = gcd(169, 273)}$.
- Step 3: $\mathbf{gcd(169, 273) = gcd(273, 169) = gcd(273 - 169, 169) = gcd(104, 169)}$.
- Step 4: $\mathbf{gcd(104, 169) = gcd(169, 104) = gcd(169 - 104, 104) = gcd(65, 104)}$.
- Step 5: $\mathbf{gcd(65, 104) = gcd(104, 65) = gcd(104 - 65, 65) = gcd(39, 65)}$.
- Step 6: $\mathbf{gcd(39, 65) = gcd(65, 39) = gcd(65 - 39, 39) = gcd(26, 39)}$.
- Step 7: $\mathbf{gcd(26, 39) = gcd(39, 26) = gcd(39 - 26, 26) = gcd(13, 26)}$.
- Step 8: $\mathbf{gcd(13, 26) = gcd(26, 13) = gcd(26 - 2 \times 13, 13) = gcd(0, 13)}$.
- Step 9: $\mathbf{gcd(0, 13) = gcd(13, 0) = 13}$.

It is known that the computational complexity of computing $\mathbf{gcd}(a, b)$ is $\mathcal{O}\big(\log \max(a, b)\big)$.
Compute $\mathbf{gcd(1538099040171999308, 15052132919125948 21)}$.

The extended Euclid's algorithm

- The Euclid's algorithm starts a sequence with a and b and proceeds by doing modular reductions on consecutive terms of the sequence until zero is reached.
- But it is possible to do more!
- Let the sequence begin with $x_0 = a$ and $x_1 = b$. At any time, let $x_k = s_k a + t_k b$. So, $s_0 = t_1 = 1$, and $s_1 = t_0 = 0$.
- The next term of the sequence is given by $x_k = x_{k-2} \bmod x_{k-1}$. Let $q_k = \left\lfloor \frac{x_{k-2}}{x_{k-1}} \right\rfloor$. Then,

$$x_k = x_{k-2} - q_k x_{k-1}, \qquad s_k = s_{k-2} - q_k s_{k-1}, \quad \text{and} \quad t_k = t_{k-2} - q_k t_{k-1}.$$

- We have to stop when $x_k = 0$, at which time $\gcd(a, b) = x_{k-1}$. But here we know more:

$$x_{k-1} = s_{k-1} a + t_{k-1} b.$$

If $\gcd(a, b) = 1$, if $x_{k-1} = 1$, this formula allows us to compute easily

$$a^{-1} \bmod b = s_{k-1} \bmod b \qquad \text{and} \qquad b^{-1} \bmod a = t_{k-1} \bmod a.$$

The extended Euclid's algorithm (example)

Goal: apply the extended Euclid's algorithm to compute $\mathbf{gcd}(77, 54)$.

- The following table illustrates the computations done by the extended Euclid's algorithm.

k	x_k	q_k	s_k	t_k
0	77		1	0
1	54		0	1
2	23	1	1	−1
3	8	2	−2	3
4	7	2	5	−7
5	1	1	−7	10
6	0	7	54	−77

- Because $x_6 = 0$, the information we seek corresponds to the row with $k = 5$. We have $\mathbf{gcd}(77, 54) = 1$, $77^{-1} \bmod 54 = -7 \bmod 54 = 47$, and $54^{-1} \bmod 77 = 10$.

The GNU MP library has a function, `mpz_gcdext`, for this; `pari-gp` also has a function, `gcdext`, for this. Let $a = 830150497265848419$ and $b = 47233264741020289 6$. Compute $a^{-1} \bmod b$ and $b^{-1} \bmod a$.

Linear maps

- When working modulo m it suffices to work with integers in the range $0, 1, \ldots, m-1$, i.e., it suffices to work with $\mathbb{Z}_m$.
- Let

$$f(x; m, a) = (ax) \bmod m$$

  be the linear map $x \mapsto (ax) \bmod m$ from $\mathbb{Z}_m$ into itself.
- Recall that a function $f(x)$ is said to be linear if $f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$ for all α, β, x, and y.
- For example, for $m = 4$ the linear map with $a = 2$ (on the left) is **not** invertible, but the linear map with $a = 3$ (on the right) is invertible.

map for $m = 4$ and $a = 2$
$0 \mapsto 0$
$1 \mapsto 2$
$2 \mapsto 0$
$3 \mapsto 2$

map for $m = 4$ and $a = 3$
$0 \mapsto 0$
$1 \mapsto 3$
$2 \mapsto 2$
$3 \mapsto 1$

Linear maps (continuation)

- Why are we interested in inverting the map? Because the map scrambles the elements of $\mathbb{Z}_m$ and we may be interested in unscrambling them (think in cryptographic terms).
- So, what is the inverse map?
- It turns out that the inverse map, if it exists, is also a linear map.
- More specifically, the inverse map of $f(x, m, a \bmod m)$ is $f(x, m, a^{-1} \bmod m)$, where $a^{-1} \bmod m$ is the modular inverse of $a \bmod m$. Indeed, if $y = f(x; m, a) = ax \bmod m$ then $x = a^{-1}y \bmod m$.
- Since the modular inverse of a modulo m only exists when $\gcd(a, m) = 1$ the linear map is invertible if and only if $\gcd(a, m) = 1$.
- Keep in mind that we wish to devise a way to encrypt information by providing public data to do so (in this case it would be m and a).
- Alas, this way of scrambling information is very easy to unscramble, so useless from a cryptography point of view.
- Modular multiplication scrambles the information but it is easy to undo if we known m and a. What about modular exponentiation?

Linear maps (a failed cryptosystem)

The Merkle-Hellman knapsack cryptosystem keeps the following information secret:

- a set $\boldsymbol{W = \{ w_1, w_2, \ldots, w_n \}}$ of $\boldsymbol{n}$ positive integers, such that $\boldsymbol{w_k}$ is a super-increasing sequence, i.e., $\boldsymbol{w_k > \sum_{i=1}^{k-1} w_i}$ for $\boldsymbol{2 \leqslant k \leqslant n}$,
- a modulus $\boldsymbol{m}$ such that $\boldsymbol{m > \sum_{i=1}^{n} w_i}$,
- a scrambling integer $\boldsymbol{a}$ such that $\boldsymbol{\gcd(a, m) = 1}$.

and publishes the following information:

- set $\boldsymbol{W' = \{w'_1, w'_2, \ldots, w'_n\}}$, where $\boldsymbol{w'_i = (a w_i) \bmod m}$, for $\boldsymbol{1 \leqslant i \leqslant n}$.

Actually, it is **much** better to publish a random permutation of $\boldsymbol{W'}$. (Homework: why?). To send a message composed by the $\boldsymbol{n}$ bits $\boldsymbol{\alpha_k}$, $\boldsymbol{1 \leqslant k \leqslant n}$, compute and send

$$\boldsymbol{C = \sum_{k=1}^{n} \alpha_k w'_k.}$$

This is a hard knapsack problem (in this case a subset sum problem). To decipher transform it into a trivial knapsack problem by computing $\boldsymbol{a^{-1} C \bmod m}$, which is equal to $\boldsymbol{\sum_{k=1}^{n} \alpha_k w_k}$ and so can be solved by a greedy algorithm.

Linear maps (Merkle-Hellman knapsack example)

The following example shows the Merkle-Hellman cryptosystem in action.

- Secret data: $\boldsymbol{W} = \{\, 1, 3, 5, 12, 22, 47 \,\}$, $m = 100$, and $a = 13$.
- Public data: $\boldsymbol{W'} = \{\, 13, 39, 65, 56, 86, 11 \,\}$,
- Unencrypted message to be sent: $\boldsymbol{A} = \{\, 0, 0, 1, 1, 0, 1 \,\}$.
- Encrypted message sent: $C = 0 \times 13 + 0 \times 39 + 1 \times 65 + 0 \times 86 + 1 \times 11 = 132$.
- To decrypt compute $132 \times 13^{-1} \bmod 100 = 32 \times 77 \bmod 100 = 64$ and then reason as follows [greedy algorithm for the easy subset sum problem]:

  1. 47 must be used to form the sum because $64 > 47$. Hence $\alpha_6 = 1$. The rest of the sum is $64 - 47 = 17$.
  2. 22 cannot be used to form the sum because $17 < 22$. Hence $\alpha_5 = 0$.
  3. 12 must be used to form the sum because $17 > 12$. Hence $\alpha_4 = 1$. The rest of the sum is $17 - 12 = 5$.
  4. As so on. In this particular case, the next iteration finishes the deciphering process.

Fermat's little theorem

- The elements of $\mathbb{Z}_m$ that have an inverse are called the **units** of $\mathbb{Z}_m$. The set containing all these units is denoted by $\mathbb{Z}_m^*$. When m is a prime number, $\mathbb{Z}_m^* = \{ 1, 2, \ldots, m-1 \}$.
- Euler's totient function $\varphi(m)$ counts how many integers in $\mathbb{Z}_m$ are relatively prime to m, i.e., it counts the number of elements of $\mathbb{Z}_m^*$. It can be computed using the formula

$$\varphi(m) = m \prod_{p|m} \left(1 - \frac{1}{p}\right),$$

  where the product is over the distinct **prime** factors of m.
- $\varphi(m)$ can be computed in `pari-gp` with the `eulerphi` function.
- Let $P = \prod_{k \in \mathbb{Z}_m^*} k$. P has to be relatively prime to m because each of its factors is relatively prime to m. [When m is prime then $P + 1 \equiv 0 \pmod{m}$ — that's Wilson's theorem — but we will not use this fact here.]
- Now assume that $a \in \mathbb{Z}_m^*$, i.e., that $\gcd(a, m) = 1$, and let us now consider what the map $f(x; m, a)$ does to the elements of $\mathbb{Z}_m^*$.
- It scrambles them! Because everything is relatively prime to m, $\mathbb{Z}_m^*$ is mapped into itself! Furthermore, it is a bijection (a one-to-one map).

Fermat's little theorem (continuation)

- So, since the map $x \mapsto ax \bmod m$ when applied to $\mathbb{Z}_m^*$ just reorders its elements, it follows that

$$Q \equiv \left(\prod_{k \in \mathbb{Z}_m^*} ak \right) \equiv \left(a^{\varphi(m)} \prod_{k \in \mathbb{Z}_m^*} k \right) \equiv (a^{\varphi(m)} P) \pmod{m},$$

  but also that (because of the reordering!)

$$Q \equiv P \pmod{m}.$$

- Since $\gcd(P, m) = 1$, $P^{-1} \bmod m$ exists, and so we can say that, for any $a \in \mathbb{Z}_m^*$, we have (this is Fermat's little theorem)

$$a^{\varphi(m)} \equiv 1 \pmod{m}.$$

- For a prime number p we have $\varphi(p) = p - 1$, and Fermat's little theorem takes the form

$$a^{p-1} \equiv 1 \pmod{p}, \qquad \text{for all } a \text{ with } \gcd(a, p) = 1.$$

  We can take care of the case $a \equiv 0 \pmod{p}$ by multiplying both sides by a:

$$a^p \equiv a \pmod{p}, \qquad \text{for all } a.$$

Fermat's little theorem (examples)

- Let's see what happens for three distinct values of m (all exponentiations are done modulo m):

$m = 7$, $e = \varphi(m) = 6$:

k	k^e	k^{e+1}
0	0	0
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6

(The values of k for which $\gcd(k, m) = 1$ have a gray background.)

$m = 10$, $e = \varphi(m) = 4$:

k	k^e	k^{e+1}
0	0	0
1	1	1
2	6	2
3	1	3
4	6	4
5	5	5
6	6	6
7	1	7
8	6	8
9	1	9

$m = 12$, $e = \varphi(m) = 4$:

k	k^e	k^{e+1}
0	0	0
1	1	1
2	4	8
3	9	3
4	4	4
5	1	5
6	0	0
7	1	7
8	4	8
9	9	9
10	4	4
11	1	11

- What happens for $m = 2 \times 3 \times 5$?
- It looks like $a^{\varphi(m)+1} \equiv a \pmod{m}$ when m does not have repeated prime factors!

Chinese remainder theorem

- Supose that you know that $x \equiv a \pmod{m}$ and that $x \equiv b \pmod{n}$.
- From the first condition x has to be equal to $a + km$ for some integer k.
- But $a + km \equiv b \pmod{n}$, and so $k \equiv m^{-1}(b - a) \pmod{n}$. The modular inverse exists for sure if $\gcd(m, n) = 1$, which we assume is the case here.
- Therefore, we know that $k = ln + c$ for some integer l, where $c = m^{-1}(b - a) \bmod n$. Note that $c = 0$ when $b = a$.
- Finaly, we get $x = a + cm + lmn$, i.e., $x \equiv a + cm \pmod{mn}$.
- It is possible to reach the same conclusion more quickly:

$$x \equiv a(n^{-1} \bmod m)n + b(m^{-1} \bmod n)m \pmod{mn}.$$

- In general, if we know that $x \equiv a_k \pmod{m_k}$, for $1 \leqslant k \leqslant K$, with the moduli m_k pairwise coprime (i.e., $\gcd(m_i, m_j) = 1$ when $i \neq j$) then, with $M = \prod_{k=1}^{K} m_k$ and $M_k = M/m_k$, we have

$$x \equiv \sum_{k=1}^{K} a_k(M_k^{-1} \bmod m_k)M_k \pmod{M}.$$

Chinese remainder theorem (problems)

Solve the following systems of congruences:

$$\begin{cases} x \equiv 0 \pmod 8 \\ x \equiv 1 \pmod 9 \end{cases}$$

$$\begin{cases} x \equiv 0 \pmod 8 \\ x \equiv 8 \pmod{16} \\ x \equiv 3 \pmod 5 \end{cases}$$

$$\begin{cases} x \equiv 2 \pmod 3 \\ x \equiv 2 \pmod 5 \\ x \equiv 2 \pmod 7 \end{cases}$$

$$\begin{cases} x \equiv 1 \pmod 2 \\ x \equiv 2 \pmod 3 \\ x \equiv 4 \pmod 5 \\ x \equiv 6 \pmod 7 \\ x \equiv 10 \pmod{11} \\ x \equiv 12 \pmod{13} \end{cases}$$

$$\begin{cases} x \equiv 12345 \pmod{2718281828} \\ x \equiv 67890 \pmod{3141592653} \end{cases}$$

- Hint: `pari-gp` groks the chinese remainder theorem (`chinese` function). For example, the first problem cn be solved in `pari-gp` by

```
chinese(Mod(0,8),Mod(1,9))
```

Fermat's little theorem (revisited)

- Let p be any prime number. By Fermat's little theorem we know that

$$x^{\varphi(p)} \equiv x^{p-1} \equiv 1 \pmod{p}, \qquad \text{when } \gcd(x,p) = 1.$$

- It follows that for any integers r and x we have

$$x^{r(p-1)+1} \equiv x \pmod{p}.$$

  For $x \equiv 0 \pmod{p}$ this is obvious. For the other cases use Fermat's little theorem to adjust the exponent.

- Now consider a second prime, q, different from p. We also have, for any integer s,

$$x^{s(q-1)+1} \equiv x \pmod{q}.$$

- Let t be the least common multiple of $p-1$ and $q-1$. If follows that

$$x^{t+1} \equiv x \pmod{p} \qquad \text{and} \qquad x^{t+1} \equiv x \pmod{q}.$$

- By the chinese remainder theorem this implies that

$$x^{t+1} \equiv x \pmod{pq}.$$

Fermat's little theorem (conclusion)

- The previous result can be generalized to K primes.
- Let $p_1, p_2, \ldots, p_K$ be K distinct primes. Here, p_1 is not necessarily the first prime (two) and so on.
- Let P be their product: $P = \prod_{k=1}^{K} p_k$.
- Let $\lambda(P)$ be the so-called Carmichael function, given by

$$\lambda(P) = \lambda(p_1 p_2 \cdots p_K) = \operatorname{lcm}(p_1 - 1, p_2 - 1, \ldots, p_K - 1).$$

- Then, for any integers k and x, we have

$$\begin{cases} x^{k\lambda(P)+1} \equiv x \pmod{P}, & \text{always,} \\ x^{\lambda(P)} \equiv 1 \pmod{P}, & \text{when } \gcd(x, P) = 1. \end{cases}$$

- This result is often presented with $\lambda(P)$ replaced by $\prod_{k=1}^{K} \varphi(p_k) = \prod_{k=1}^{K} (p_k - 1)$. The later is a multiple of the former.
- This means that in a modular exponentiation we may usually reduce the exponent modulo $\lambda(P)$. Exponents that are multiples of $\lambda(P)$ are the exception; they have to be handled separately.

Modular exponentiation

- The modular exponentiation $a^b \bmod m$ can be done recursively using the following two observations:

$$a^{2n} \bmod m = (a^2)^n \bmod m \qquad \text{and} \qquad a^{2n+1} \bmod m = a(a^2)^n \bmod m.$$

  If follows that it can be done using $\mathcal{O}(\log n)$ modular multiplications.

- Example:

$$\begin{aligned}
&13^{21} \bmod 71 = 13 \times (13^2)^{10} \bmod 71 = 13 \times 27^{10} \bmod 71,\\
&27^{10} \bmod 71 = (27^2)^5 \bmod 71 = 19^5 \bmod 71,\\
&19^5 \bmod 71 = 19 \times (19^2)^2 \bmod 71 = 19 \times 6^2 \bmod 71,\\
&6^2 \bmod 71 = 36 \bmod 71,
\end{aligned}$$

  backsubstituting...

$$\begin{aligned}
&19^5 \bmod 71 = 19 \times 36 \bmod 71 = 45 \bmod 71,\\
&27^{10} \bmod 71 = 45 \bmod 71,\\
&13 \times 27^{10} \bmod 71 = 17 \bmod 71,\\
&13^{21} \bmod 71 = 17 \bmod 71 = 17.
\end{aligned}$$

Modular exponentiation (another way)

- Let the exponent n, with $N+1$ bits, be represented in base-2 as follows:

$$n = \sum_{k=0}^{N} n_k 2^k.$$

- Then,

$$a^n \bmod m = a^{\sum_{k=0}^{N} n_k 2^k} \bmod m = \prod_{k=0}^{N} a^{n_k 2^k} \bmod m.$$

- Using the example of the previous slide, we have $n = 21 = 10101_2$, so $N = 4$. Thus,

k	a^{2^k}	use in the final product?
0	13	yes
1	27	no; note that $27 = 13^2 \bmod 71$
2	19	yes; note that $19 = 27^2 \bmod 71$
3	6	no; in general, each number is the square of the previous number
4	26	yes

So, $13^{21} \bmod 71 = 13 \times 19 \times 26 \bmod 71 = 17$.

- Compute $12345^{67890} \bmod 123456789$.

Modular exponentiation (a slightly better way)

- It is possible to do slightly better (Brauer's algorithm). Let the exponent n, with $d+1$ base-B digits, be represented in base-B as follows:

$$n = \sum_{k=0}^{d} n_k B^k = n_0 + B\Big(n_1 + B\big(n_2 + B(\ldots + n_d)\big)\Big).$$

  The last equality is the Horner's rule to evaluate a polynomial. Note that $0 \leqslant n_k < B$. (Usually, B is a power of 2.)

- Then, $a^n \bmod m$ can be evaluated using the following sequence of steps:

$$\begin{array}{llll} r_0 = a^{n_d} \bmod m & r_1 = r_0^B & r_2 = a^{n_{d-1}} r_1 \bmod m & r_3 = r_2^B \\ r_4 = a^{n_{d-2}} r_3 \bmod m & r_5 = r_4^B & \cdots & \cdots \\ r_{2d} = r_{2d-1} a^{n_0} \bmod m & & & \end{array}$$

- When $B = 8$, the 8 possible values of $a^{n_k} \bmod m$ can be precomputed and stored — **in an interleaved way** to avoid side-channel attacks — in memory.

first word of a^0	first word of a^1	first word of a^2	first word of a^3	first word of a^4	first word of a^5	first word of a^6	first word of a^7
second word of a^0	second word of a^1	second word of a^2	second word of a^3	second word of a^4	second word of a^5	second word of a^6	second word of a^7
...	...	...	...	...	...	...	...

- To explore further: addition chains.

Multiplicative order

- Fermat's little theorem says that $x^{\lambda(m)} \equiv 1 \pmod{p}$ for any $x \in \mathbb{Z}_m^*$.
- For a given $x \in \mathbb{Z}_m^*$ what is the least exponent o such that $x^o \bmod m = 1$?
- This least exponent is called the order of x modulo m (the function `znorder` computes this in `pari-gp`).
- The order **has** to be a divisor of $\lambda(m)$.
- For a prime number p, $\lambda(p) = \varphi(p) = p - 1$.
- It turns out that there are $\varphi(p-1)$ elements of $\mathbb{Z}_p^*$ with maximal order $p-1$. These elements are called **primitive roots**.
- `pari-gp` has a function, `znprimroot`, to compute one of them.
- They generate $\mathbb{Z}_p^*$ multiplicatively. In particular, let r be one primitive root. Then, for $k = 0, 1, 2, \ldots, p-2$, $r^k \bmod p$ takes all values of $\mathbb{Z}_p^*$ (without repetitions).
- We can therefore speak of logarithms (modulo p), with respect to base r. The logarithm of $a = r^x \bmod p$ in base r is obviously x. This so-called discrete logarithm problem is currently very hard to solve when p is large.

Primality tests

- One way to prove that a given number m is prime is to find one of its primitive roots.
- Choose a random a between 2 and $m - 2$.
- If $gcd(a, m) \neq 1$, then m is not prime. Better yet, the greatest common divisor allow us to partially factor m.
- By Fermat's little theorem we know that $a^{m-1} \equiv 1 \bmod m$. If this is not so, then definitely m is not prime.
- Furthermore, when m is an odd number, we must have either $a^{(m-1)/2} \equiv 1 \bmod m$ or $a^{(m-1)/2} \equiv -1 \bmod m$.
- Now, it can be shown that a is a primitive root modulo m if, for every prime divisor d of $m - 1$, we have $a^{(m-1)/d} \bmod m \neq 1$. If a satisfies these conditions then the order of a modulo m must be $m - 1$, and thus m must be prime.
- If not, try another a.
- These exist composite numbers, called Carmichael numbers, for which $a^{m-1} \bmod m = 1$ for all a which are relatively prime to m. For these numbers, $\lambda(m) \mid (m - 1)$.

The Miller-Rabin primality test

- Goal: to test if the odd number n, with $n > 3$, is a prime number or not.
- Result of the test: either n is definitely not prime or it may be prime (a probable prime number); in the second case, the probability that the test fails to identify a composite number is **at most** 0.25.
- How it is done (to increase the confidence on the result, do steps 2 to 6 several times):
  1. Let $n - 1$ be written as $n - 1 = 2^r d + 1$, with d an odd number (so r is as large as possible).
  2. Select at random an integer a uniformly distributed in the interval $2 \leqslant a \leqslant n - 2$.
  3. If $\gcd(a, n) \neq 1$, then n is definitely a composite number.
  4. Compute $x_0 = a^d \bmod n$. If $x_0 = 1$ or $x_0 = n - 1$ then n is a probable prime.
  5. Otherwise, for $k = 1, 2, \ldots, d - 1$, compute $x_k = x_{k-1}^2 \bmod n$. If $x_k = n - 1$ then n is a probable prime.
  6. Finally, if we get here, say that n is definitely a composite number (because Fermat's little theorem failed because $a^{(m-1)/2} \equiv \pm 1 \bmod m$).
- The composite numbers that pass this test (meaning that the algorithm above says that they are probable primes) for a given a are called base-a strong pseudo-primes.

The Diffie-Hellman key exchange protocol

- Alice and Bob have never met but wish to exchange a secret key (perhaps for a symmetrical cipher algorithm).
- They agree, on a public channel, on a prime p and on primitive root r modulo p. (Choose a prime number with an easy to find factorization of $p-1$, say a Sophie Germain prime.)
- Alice generates a random number α between 2 and $p-2$, or, better yet, between $p^{0.8}$ and $p-p^{0.8}$, and sends Bob the integer $A = r^{\alpha} \bmod p$. She keeps α only to herself.
- Likewise, Bob generates a random number β between 2 and $p-2$, and sends Alice the integer $B = r^{\beta} \bmod p$. He keeps β only to himself.
- Alice computes $S = B^{\alpha} \bmod p = r^{\alpha\beta} \bmod p$.
- Bob computes $S = A^{\beta} \bmod p = r^{\alpha\beta} \bmod p$.
- They have arrived at the same number, which is their shared secret. They can now discard A, B, α, β, and p (**do not** reuse p many times).
- Anyone eavesdropping their communications (Eve?, Mallory?) has to either infer α from A or β from B. This is known as the discrete logarithm problem, which is currently a very hard problem to solve.

Diffie-Hellman key exchange protocol (exercises)

- Let $p = 101$ and $r = 2$. Alice chooses $\alpha = 52$, and so $A = 97$. Bob chooses $\beta = 46$ and so $B = 82$. Confirm that the common secret is $S = 58$.
- Let $p = 3141601$ and $r = 26$. Alice chooses $\alpha = 2437429$, and so $A = 1282989$. Bob chooses $\beta = 2988228$ and so $B = 2426580$. The common secret is $S = 1669355$. Try to find α given A and to find β given B.
- Let $p = 31415926541$ and $r = 10$. Alice chooses $\alpha = 29770170945$, and so $A = 5728872032$. Bob chooses $\beta = 23956179675$ and so $B = 22727460975$. The common secret is $S = 26991399064$. Try to find α given A and to find β given B.
- Let $p = 31415926535897239$ and $r = 6$. Alice chooses

  $$\alpha = 2459372999633886947, \qquad \text{and so} \qquad A = 240813023655276871 6.$$

  Bob chooses

  $$\beta = 2502449096145193611, \qquad \text{and so} \qquad B = 434542471090467423.$$

  The common secret is $S = 126722235922685222 8$. Can you find by yourself α given A and β given B? Hint: `pari-gp` does this with the `znlog` function.

Diffie-Hellman key exchange protocol (man-in-the-middle attack)

- Mallory, being a powerful individual, can intercept and replace all messages between Alice and Bob.
- Here's how he can compromise the Diffie-Hellman key exchange protocol.
- Mallory intercepts all messages coming from Alice in the Diffie-Hellman key exchange protocol and impersonates Bob. At the end of the key-exchange protocol he will share a secret key with Alice.
- Likewise, Mallory intercepts all messages coming from Bob in the Diffie-Hellman key exchange protocol and impersonates Alice. At the end of the key-exchange protocol he will share a secret key with Bob (different from the one he shares with Alice).
- From this point on, he decrypts all messages between them, using the appropriate shared secret key, and reencrypts them using the other shared secret key. He may even modify the messages.
- But Alice and Bob can counter this if they send their messages in two or more distinct parts in an interlocked fashion (this assumes that decoding can only be performed after all parts have been received).

ElGamal public key cryptosystem

- Alice and Bob agree on a large prime number p and on an element g of $\mathbb{F}_p^*$ with a large prime order
- Alice chooses a private key a, with $1 < a < p - 1$, and publishes $A = g^a \bmod p$.
- Bob chooses a random ephemeral key k.
- He uses Alice's public key A to compute $c_1 = g^k \bmod p$ and $c_2 = mA^k \bmod p$, where m is the plaintext.
- He then sends (c_1, c_2) to Alice.
- To recover the plaintext m, Alice computes $m = (c_1^a)^{-1} c_2 \bmod p$. This works because $(c_1^a)^{-1} c_2 = g^{-ak} m g^{ak} = m \bmod p$.
- An eavesdropper has to find k from c_1 (discrete logarithm problem).
- A middle-man can easily manipulate c_2; for example, to replace m by $2m$ all that is necessary is to replace c_2 by $2c_2 \bmod p$.
- This public key cryptosystem, implemented exactly as above, has some security problems.

The Rivest-Shamir-Adleman cryptosystem

- The Rivest-Shamir-Adleman cryptosystem (or RSA for short) is based on the observation (Fermat's little theorem) that when N is the product of two distinct prime numbers, i.e., $N = pq$, then for any x and any k we have

$$x^{k\lambda(N)+1} \equiv x \pmod{N}.$$

- In particular, the transformation

$$y = x^e \bmod N$$

can be undone using the transformation

$$x = y^d \bmod N$$

provided that

$$ed \equiv 1 \pmod{\lambda(N)},$$

i.e., provided that $e = d^{-1} \bmod \lambda(N)$.

Rivest-Shamir-Adleman cryptosystem (continuation)

- The key observation is that this is easy to do only when $\lambda(N)$ is known.
- In turn, $\lambda(N)$ can be computed easily only when the factorization of N is known: $\lambda(N) = \lambda(pq) = \mathrm{lcm}(p-1, q-1)$.
- Since the factorization of a large number is considered to be a hard problem, given N and e it is hard to compute d, and thus to recover y given x.
- It is thus possible to publish N and e without revealing too much information.
- So, anyone using the RSA public key cryptosystem **publishes** hers/his own N and e.
- Sending a ciphered message to someone entails using that person's public modulus (N) and exponent (e)
- About the choice of the primes p and q:
  1. They should be random (**do not** reuse primes!)
  2. $p-1$ and $q-1$ should not have small prime factors

Rivest-Shamir-Adleman cryptosystem (continuation)

- Alice wants to send a message M to Bob.
- First, she fetches Bob's public encription data: a modulus N_{bob} and an encryption exponent e_{bob}.
- Then, she computes the ciphered message $C = M^{e_{\text{bob}}} \bmod N_{\text{bob}}$, and sends it to Bob.
- Bob knows that $N_{\text{bob}} = p_{\text{bob}} q_{\text{bob}}$ (the secret information that only he knows), and so he can compute d_{bob}, the decryption exponent, such that $e_{\text{bob}} d_{\text{bob}} \equiv 1 \pmod{\lambda(N_{\text{bob}})}$.
- Using d_{bob} he can decipher C: $M = C^{d_{\text{bob}}} \bmod N_{\text{bob}}$.
- This works because

$$C^{d_{\text{bob}}} \bmod N_{\text{bob}} = M^{e_{\text{bob}} d_{\text{bob}}} \bmod N_{\text{bob}} = M^{k\lambda(N_{\text{bob}})+1} \bmod N_{\text{bob}} = M$$

- Note that the decryption can be done more efficiently using the Chinese remainder theorem (instead of doing one modular exponentiation modulo N do two modular exponentiations, one modulo p and another modulo q, and at the end combine them using the Chinese remainder theorem) — but, be aware of side-channel attacks...

Rivest-Shamir-Adleman cryptosystem (conclusion)

- The RSA cryptosystem can do even more: it is possible to ensure that the message came from a specified sender (that makes virtually impossible to forge a properly signed message)
- Main idea: Alice computes a message digest (hash) S of the message she wants to send to Bob and enciphers it using her own modulus and **private** decryption exponent:

$$S_{\text{alice}} = S^{d_{\text{alice}}} \bmod N_{\text{alice}}.$$

- Bob can recover S using Alice's **public** data:

$$S_{\text{alice}}^{e_{\text{alice}}} \bmod N_{\text{alice}} = S^{e_{\text{alice}} d_{\text{alice}}} \bmod N_{\text{alice}} = S^{k\lambda(N_{\text{alice}})+1} \bmod N_{\text{alice}} = S$$

- So, Bob decodes the message Alice sent him, computes its message digest, and compares it with the S obtained from the S_{alice} data. If they match it is almost certain that it was indeed Alice that has sent the message. Otherwise, someone else was trying to impersonate Alice.

Rivest-Shamir-Adleman cryptosystem (big example)

In August 1977, in his Scientific American Mathematical games column, Martin Gardner posed the following RSA challenge.

- Character encoding: space is 00, `A` to `Z` are 01 to 26. Other two digits combinations are illegal.
- The plain text is obtained by concatenating the two digits of each character encoding; the result is a large base-10 integer M.
- The plain text was then encoded using the modulus

$$N = 114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541$$

and the exponent $e = 9007$. The encoded message is $C = M^e \bmod N$, with

$$C = 968696137546220614771409222543558829057599911245743198746951209308162982251457083569314766228839896280133919905518299451578151541.$$

- What is M?

Rivest-Shamir-Adleman cryptosystem (solution of the big example)

- It took more than 10 years until $N = pq$ was factored (in 1977 it was estimated that the factorization would take much more time!):

  $$p = 3490529510847650949147849619903898133417764638493387843990820577$$

  and

  $$q = 32769132993266709549961988190834461413177642967992942539798288533.$$

- That made possible the computation of $d = e^{-1} \bmod \operatorname{lcm}(p-1, q-1)$;

  $$d = 20912395050161373690941936346810195773046184093006090879304842322045608569697121472257875853682203172258717888678557376735780271.$$

- Once d was known, M was recovered from $M = C^d \bmod N$:

  $$M = 20080500130107090300231518041900011805001917210501130919080015191909061801070 5.$$

- $20 \to T$, $08 \to H$, $05 \to E$, and so on. (The complete decryption is in the first slide.)

- Any exponent of the form $d + k \operatorname{lcm}(p-1, q-1)$ works. Try a few values of k to find the exponent with the smallest sideways addition (population count).

Finite fields

- It is now time to generalize the modular arithmetic concept.
- In the so-called **finite fields** we do arithmetic on integers modulo a **prime** number p **and** we work with **polynomials** with coefficients in $\mathbb{Z}_p$.
- There is one extra twist: we also work modulo a polynomial!
- So our modular arithmetic will have two different aspects:
  - all integer arithmetic is done modulo a prime number p, and
  - all polynomial arithmetic is done modulo a polynomial of degree d.
- Not all polynomials of degree d can be used as the modulus: only those that are irreducible can be used. Just like a prime number, a polynomial is irreducible modulo p if it is not possible to factor it modulo p.
- The irreducibility of the polynomial modulus is fundamental. It ensures that the only polynomial of degree smaller than that of the modulus polynomial that does not have an inverse is the zero polynomial (and that is a fundamental property of a field).

Finite fields (more info)

- The modulus polynomial can (and should) be a **monic** polynomial; the leading coefficient of a monic polynomial is one.
- Indeed, let $P(x)$ be the modulus polynomial, and let $A(x)$ be any polynomial. Then $A(x) \bmod P(x)$ is the remainder $R(x)$ of the division of $A(x)$ by $P(x)$. We have $A(x) = Q(x)P(x) + R(x)$, where $Q(x)$ is the quotient:

$$\begin{array}{l|l} A(x) & P(x) \\ \hline R(x) & Q(x) \end{array}$$

- Now, if we replace $P(x)$ by $\alpha P(x)$, where α belongs to $\mathbb{Z}_p^*$ — recall that all integer arithmetic is done modulo p and that all elements of $\mathbb{Z}_p^*$ are invertible — then we have $A(x) = \left(\alpha^{-1}Q(x)\right)\left(\alpha P(x)\right) + R(x)$, so the remainder is the case no matter how α was selected.
- When the (irreducible) modulus polynomial has degree k the finite field is usually denoted by $\mathbb{F}_{p^k}$ or by $\mathbf{GF}(p^k)$; in publications involving finite fields, p^k is often replaced by the easier to write q (if so, q has to be the power of a prime).
- For the particular case $k = 1$ we have that $\mathbb{F}_p$ is the same as $\mathbb{Z}_p$.

Finite fields (example)

- Let us work with the prime $p = 5$.
- Let us work with the irreducible polynomial (modulo 5):

  $$P(x) = x^3 + x^2 + 3x + 4.$$

  This irreducible polynomial was found using the following `pari-gp` code:

  ```
  x = ffgen([5,3]);
  x.mod
  ```

  (`x.p` gives the integer modulus, in this case 5).
- Each element of the finite field $\mathbb{F}(5^3)$ is a polynomial of the form

  $$a_2x^2 + a_1x + a_0$$

  where $a_0, a_1, a_2 \in \mathbb{F}_5$.
- Addition and subtraction of polynomials is done in the usual way (modulo 5).
- Multiplications is done in the usual way, but replacing x^3 by $-x^2 - 3x - 4$, i.e., by $4x^2 + 2x + 1$. (Why?)

Finite fields (example)

- Continuing the example of the previous slide, the quotient of the division of x^3 by $P(x)$ is $Q(x) = 1$, and so $x^3 \bmod P(x) = x^3 - P(x) = -x^2 - 3x - 4 = 4x^2 + 2x + 1$.
- In pari-gp, this can be confirmed by doing

```
x^3
```

- Here is a larger example:

$$
\begin{array}{rrrrrrr|rrrr}
 & 1x^5 & 4x^4 & 3x^3 & 0x^2 & 1x^1 & 3x^0 & 1x^3 & 1x^2 & 3x^1 & 4x^0 \\
\cline{8-11}
- & 1x^5 & 1x^4 & 3x^3 & 4x^2 & & & 1x^2 & 3x^1 & 2x^0 & \\
\cline{1-4}
 & 0x^5 & 3x^4 & 0x^3 & 1x^2 & & & & & & \\
 & - & 3x^4 & 3x^3 & 4x^2 & 2x^1 & & & & & \\
\cline{3-6}
 & & 0x^4 & 2x^3 & 2x^2 & 4x^1 & & & & & \\
 & & - & 2x^3 & 2x^2 & 1x^1 & 3x^0 & & & & \\
\cline{4-7}
 & & & 0x^3 & 0x^2 & 3x^1 & 0x^0 & & & &
\end{array}
$$

pari-gp confirmation (the modulo arithmetic is done automatically):

```
x^5+4*x^4+3*x^3+x+3
```

Finite fields (useful algorithms)

- Euclid's algorithm works!
- In particular, the extended Euclid's algorithm can be used to compute inverses.
- The modular exponentiation algorithm also works.
- Since in the finite field $\mathbb{F}_q$ — recall that $q = p^k$ — we have

  $$a^q = a \qquad \text{for all } a \in \mathbb{F}_q$$

  (this is similar to Fermat's little theorem), the inverse can also be computed using

  $$a^{-1} = a^{q-2}$$

- Note that the exponents can be reduced modulo $q - 1$.
- The factorization of $q - 1$ is something that is useful to know.

Finite fields (primitive elements)

- The invertible elements (the units) of $\mathbb{F}_q$ form a set, denoted by $\mathbb{F}_q^*$.
- For a finite field we have $\mathbb{F}_q^* = \mathbb{F}_q \backslash \{0\}$.
- The order of an element of $\mathbb{F}_q^*$ is the smallest exponent o for which $a^o = 1$.
- The order has to divide $q-1$, which is the number of elements of $\mathbb{F}_q^*$.
- A primitive element has maximal order.
- Repeated multiplication by the same primitive element **generates** $\mathbb{F}_q^*$; when that happens the field is said to be a multiplicative cyclic group.
- There exist $\varphi(q-1)$ primitive elements: if r is a primitive element then r^e will also be a primitive element if and only if $\gcd(e, q-1) = 1$.
- Therefore, there exist lots of primitive elements if q is large, so finding one is easy (if the factorization of $q-1$ is known, see next slide).
- In `pari-gp` we can compute a primitive element using the function `ffprimroot()`.

Finite fields (one way to find a primitive polynomial)

Just like in standard modular arithmetic, r is a primitive element of $\mathbb{F}_q^*$ if and only if

- $r^{q-1} = 1$, and
- for every prime divisor d of $q-1$, we have $r^{(q-1)/d} \neq 1$.

One way to find an irreducible polynomial is to find one of the primitive roots of the finite field it generates. So, to compute an irreducible polynomial of degree k when we are working modulo p — finite field $\mathbb{F}_q$ with $q = p^k$ — do the following:

1. Choose a monic polynomial of degree k.
2. Choose a desired primitive element r, say, $r = x$ (that choice is particularly useful, read the slide about cyclic redundancy checksums).
3. Check if r is a primitive element.
4. If so, the polynomial is irreducible, and we are done.
5. If not, then the polynomial may be irreducible but r is not a primitive element or it is not irreducible; go back to the beginning and try another polynomial.
6. Since there exist $\varphi(q-1)$ primitive elements when the polynomial is irreducible, and there exist $\frac{1}{k}\sum_{d|k}\mu(d)p^d$ irreducible polynomials of degree k modulo p, this procedure finds one of them in a reasonable amount of time.

Applications of finite fields

The Diffie-Hellman key exchange protocol can be trivially extended to finite fields.

- Unique difference: Alice and Bob, instead of agreeing on a prime and on one of its primitive roots, have to agree on a finite field (prime p, irreducible monic polynomial of degree k) and on one of its primitive elements.
- Anyone wishing to infer the shared secret has to solve the discrete logarithm problem, in this case for finite fields.

Elliptic curves (discussed later in this course) also work in finite fields.

Shamir's secret sharing scheme also works in finite fields (discussed later in this course).

- Unique difference: the coefficients of the polynomials, instead of belonging to $\mathbb{F}_p$, belong to $\mathbb{F}_{p^k}$. Nice! Here we have polynomials whose coefficients are other polynomials (in another variable and subjected to modulo arithmetic in two distinct ways!)

Finite fields and Cyclic Redundancy Chechsums (CRCs)

- The so-called cyclic redundancy checksum (CRC) is a way to compute a "signature" of a data set (used at the hardware level as a simple way to perform error detection).
- The data is transformed into a polynomial, and the CRC is just the remainder of that polynomial when divided by a known polynomial.
- Usually, the modulus polynomial is an irreducible polynomial having x as one of its primitive elements (a so-called primitive polynomial).
- Furthermore, this is done with $p = 2$, i.e., in the finite field $\mathbb{F}_{2^k}$. This is so because in the base field, $\mathbb{F}_2 = \mathbb{Z}_2$, addition and multiplication are particularly simple: addition is the **exclusive-or** binary logic operator and multiplication is the **and** binary logic operator.
- They are not useful (i.e., unsafe) in cryptographic applications as a way to compute message hashes (due to its linear nature, it is trivial to forge a message having a specific message hash).
- But it can be used as a hash function in an implementation of a hash table.

Elliptic curves

- Now we get to play with some weird stuff.
- We will do addition is a strange way.
- The addition operator $+$ is a binary operator; it takes two elements of a group and produces a third element of the same group.
- Addition properties (in any group):

  commutative law $x + y = y + x$

  associative law $x + (y + z) = (x + y) + z$

- Idea: suppose we have a plane curve with the following property: any straight line intersects it in exactly three points, counted with multiplicity. If so, the addition of two points on that line can be the third point!
- To make this work, it is necessary to treat the point at infinity as a legitimate point (use homogeneous coordinates, also known as projective coordinates).
- Three intersection points $\Rightarrow$ **cubic equation**.

Elliptic curves (cubic equation)

- The cubic equations we will consider have in the following form (Weierstass parameterization):

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

- Both x and y belong to a field F (or are the point at infinity).
- With a change of variables (which in some cases cannot be done due to divisions by zero), the equation above can be put in the so-called Weierstrass form

(*) $y^2 = x^3 + ax + b.$

- The so-called discriminant of the curve E, whose points satisfy equation (*), is the quantity

$$\Delta(E) = -16(4a^3 + 27b^2).$$

To avoid degenerate curves this discriminant cannot be zero.

Elliptic curves (homogeneous coordinates)

- In homogeneous coordinates we add a third coordinate: z.
- (x, y) becomes (X, Y, Z).
- (X, Y, Z), for any $Z \neq 0$, corresponds to the two-dimensional point $\left(\frac{X}{Z}, \frac{Y}{Z}\right)$ [it's an equivalence class].
- $Z = 0$ represents the "points at infinity"; X and Y then specify the direction.
- For an elliptic curve in Weierstrass form, $y^2 = x^3 + ax + b$, for very large x we have $y \approx \pm x^{3/2}$.
- So, very far from the origin, y will be considerably larger than x.
- The homogeneous coordinates of the point at infinity (there are two but only one gets to be used) are $(0, 1, 0)$.

Elliptic curves (pari-gp)

- In pari-gp, the general curve

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

can be specified using the command

```
E=ellinit([a1,a2,a3,a4,a6]);
```

- In pari-gp, the special curve

$$y^2 = x^3 + ax + b$$

can obviously be specified using the command

```
E=ellinit([0,0,0,a,b]);
```

The shortcut

```
E=ellinit([a,b]);
```

can also be used.

Elliptic curves over finite fields (pari-gp)

- In pari-gp it is also possible to specify the **field** over which all computations will be performed.
- This is specified in a second (optional) argument to `ellinit`.
- If this second argument
  - ⋆ is missing or is the integer $\mathbf{1}$, the field will be $\mathbb{Q}$
  - ⋆ is the integer p, a prime number, or is a `Mod(*,p)`, the field will be $\mathbb{F}_p$
  - ⋆ is the value returned by `ffgen([p,k])`, the field will be the finite field $\mathbb{F}_{p^k}$
  - ⋆ is a real number, the field will be $\mathbb{C}$

  It may also be a more exotic object.
- The number of points on the elliptic curve can be computed using the `ellcard` function.
- In the specific case of the field $\mathbb{F}_p$ the number of points on the elliptic curve can also be computed using the more efficient `ellsea` function.

Playing with elliptic curves (pari-gp)

- To get a list of official pari-gp tutorials consult this web page.
- In particular, read the elliptic curves tutorial.
- Better yet, read the elliptic curves over finite fields tutorial.
- You can also look at the list of functions related to elliptic curves.
- Let's play!

```
/* find an elliptic curve of the form y^2=x^3+x+1 */
/* over Fp which has a prime number of points     */
forprime(p=2^100,oo,E=ellinit([1,1],p);\
         q=ellsea(E);if(isprime(q),break;););
E.p              /* print the modulus                          */
q=ellsea(E)      /* print the number of points of the curve */
G=E.gen;G=G[1] /* get a generator (there is only one)      */
ellorder(E,G)
```

Elliptic curves (adding two points — geometric interpretation)

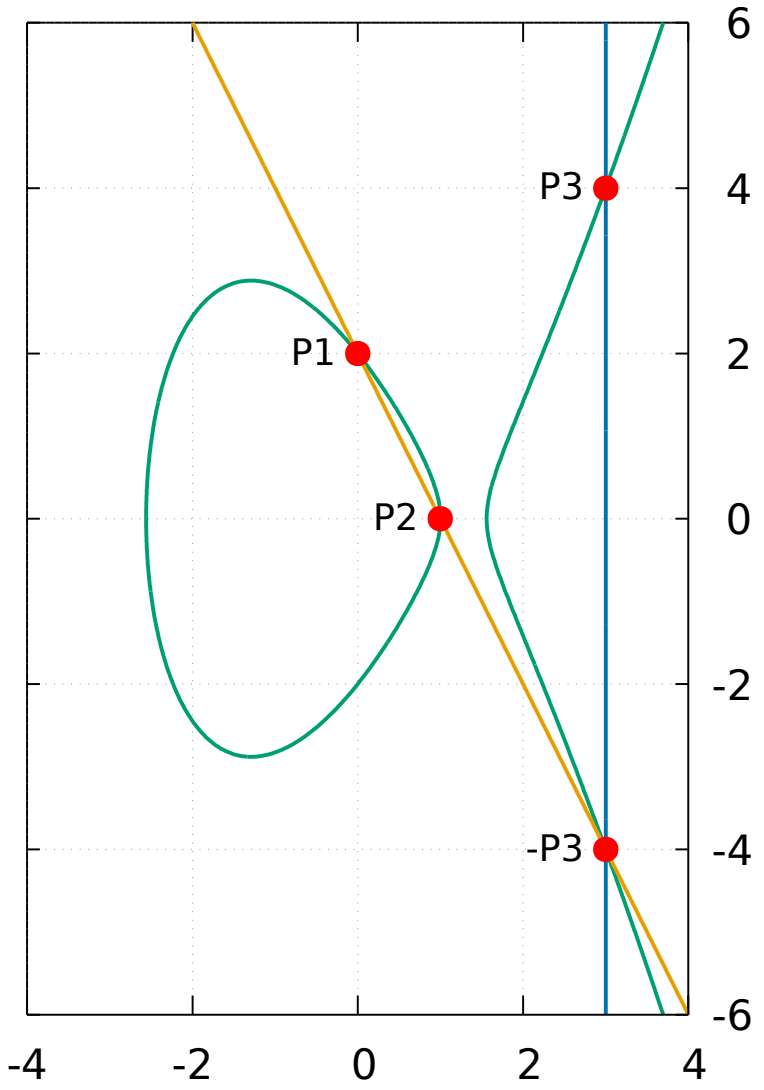
- elliptic curve over $\mathbb{Q}$:

$$y^2 = x^3 - 5x + 4.$$

- pari-gp code:

```
E=ellinit([0,0,0,-5,4]);
P1=[0,2];
P2=[1,0];
ellisoncurve(E,P1)
ellisoncurve(E,P2)
P3=elladd(E,P1,P2);
```

- The point of infinity is the **neutral** element (the zero).
- Adding to a point the point at infinity leaves it unchanged.
- Adding a point to its symmetric (its reflection on the x axis) gives rise to the point at infinity.



Elliptic curves (adding two points — some formulas)

- The equation of a straight line that passes through two distinct points (x_1, y_1) and (x_2, y_2) is

$$(x - x_1)(y_2 - y_1) = (x_2 - x_1)(y - y_1).$$

- It can be put in the form $Ax + By + C = 0$.
- If the inverse of B exists (i.e., the line is not a **vertical** line), then we can say that

$$y = Dx + E.$$

- Putting this in the cubic equation $y^2 = x^3 + ax + b$ gives rise to a polynomial equation of third degree in x of the general form

$$x^3 + \alpha x^2 + \beta c + \gamma = 0.$$

- It has three solutions. Two of them must be x_1 and x_2. The third one is the x coordinate of the point we are looking for.
- When we are working with rational numbers ($\mathbb{Q}$) because the sum of the roots is $-\alpha$ it follows that this third root must also be a rational number!

Elliptic curves (adding the same point — geometric interpretation)

- elliptic curve over $\mathbb{Q}$:

$$y^2 = x^3 - 5x + 4.$$

- pari-gp code:

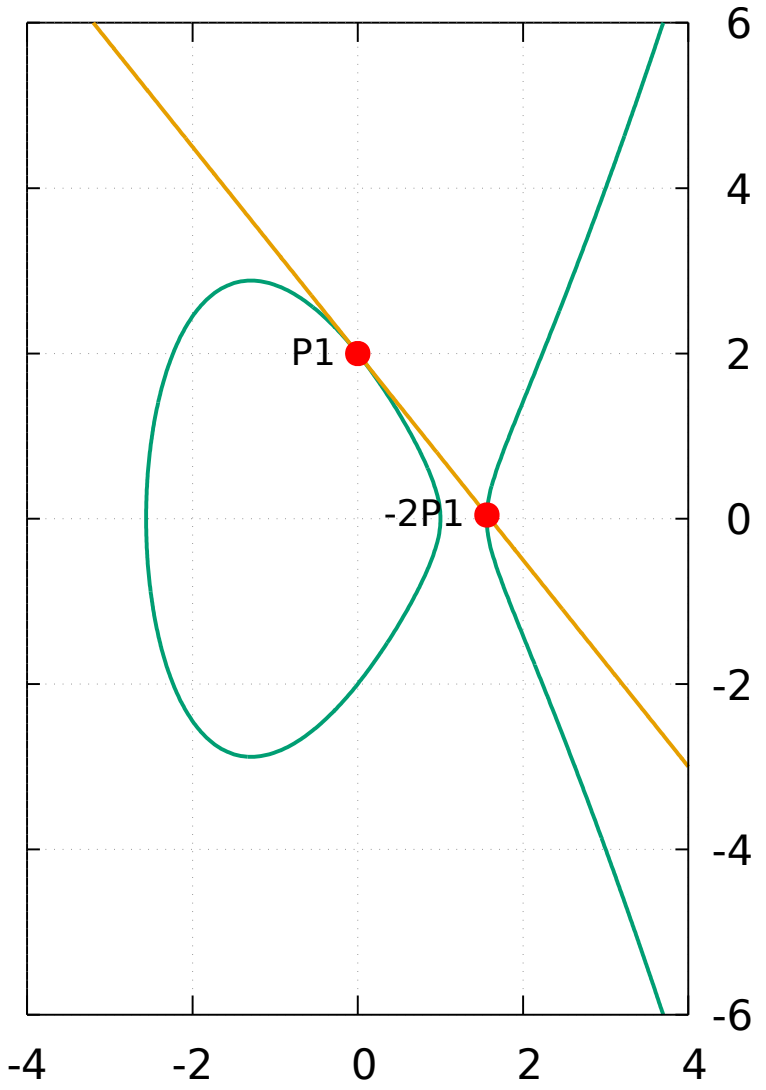
```
E=ellinit([0,0,0,-5,4]);
P1=[0,2];
ellisoncurve(E,P1)
P2=ellmul(E,P1,2);
/* same as P2=elladd(E,P1,P1); */
```

We have

$$2P1 = \left(\frac{25}{16}, \frac{-3}{64}\right)$$

$$3P1 = \left(\frac{96}{625}, \frac{-28106}{15625}\right)$$

$$4P1 = \left(\frac{352225}{576}, \frac{209039023}{13824}\right)$$



Multiplication by an integer (adding a point to itself several times)

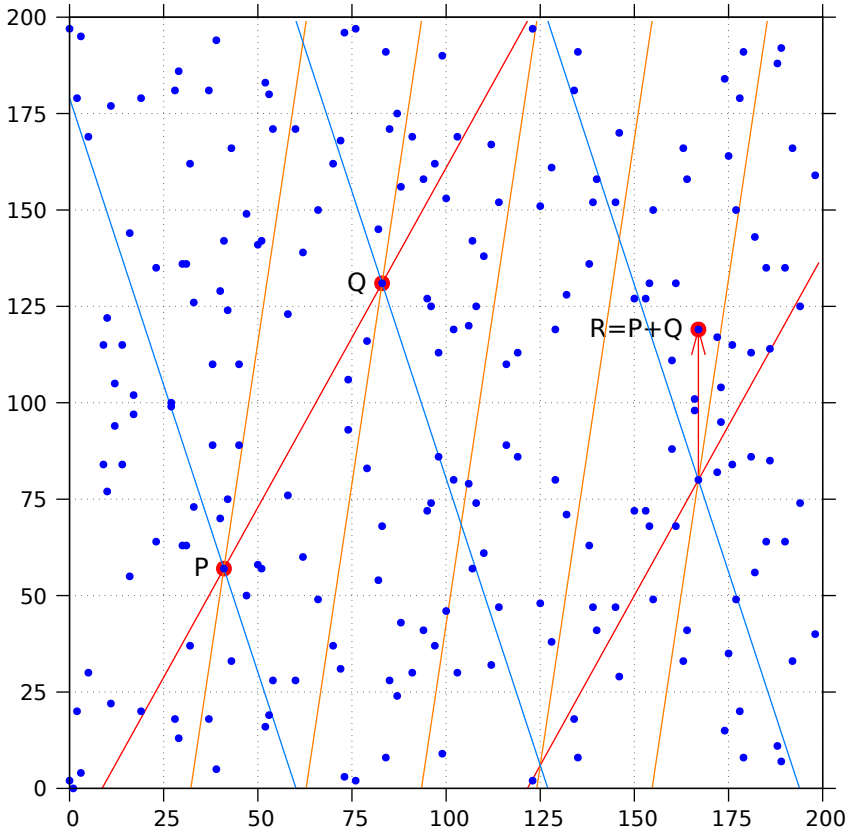
- We can now define the mathematical operation that is useful for cryptographic purposes: multiplication of a point by an integer.
- This corresponds to adding a point with itself several times.
- In terms of cryptographic applications this corresponds roughly to the modular exponentiation done in finite fields.
- Example: to compute, say, $\boldsymbol{11P}$ we can proceed as follows:
  1. $\boldsymbol{11 = 1 + 2 + 8}$
  2. compute $\boldsymbol{2P = P + P}$
  3. compute $\boldsymbol{4P = (2P) + (2P)}$
  4. compute $\boldsymbol{8P = (4P) + (4P)}$
  5. finally, compute $\boldsymbol{11P = (1P) + (2P) + (8P)}$
- This multiplication algorithm is similar in spirit to the algorithm presented in the modular exponentiation slides.
- Hard problem (on some elliptic curves): given $\boldsymbol{P}$ and $\boldsymbol{kP}$ find $\boldsymbol{k}$.

Elliptic curves (aspect of the "curve" on a finite field)

- elliptic curve over $\mathbb{F}_{199}$:

$$y^2 = x^3 - 5x + 4.$$

- it has 218 points (including the point "at infinity"):




```
p=199;
E=ellinit([0,0,0,-5,4],p);
N=ellsea(E)          /* 218 */
P=Mod([41,57],p);
ellisoncurve(E,P)    /* 1   */
Q=Mod([83,131],p);
ellisoncurve(E,Q)    /* 1   */
R=elladd(E,P,Q);
lift(Q[1])           /* 167 */
lift(Q[2])           /* 119 */
```

Diffie-Hellman using elliptic curves

- We can now explain how the Diffie-Hellman secret sharing scheme can be done using elliptic curves.
- Alice and Bob agree on an elliptic curve and on a point $\boldsymbol{P}$ — of large order — of that elliptic curve.
- Alice chooses a private random integer $\boldsymbol{k_A}$ and sends $\boldsymbol{k_A P}$ to Bob.
- Bob chooses a private random number $\boldsymbol{k_B}$ and sends $\boldsymbol{k_B P}$ to Alice.
- The shared secret $\boldsymbol{S}$ is the point $\boldsymbol{k_A k_B P}$; Alice and Bob can compute it easily using the private information they have and the information each received from the other one.
- A third party will have to attempt to compute $\boldsymbol{k_A}$ from the information Alice sent to Bob (over a possibly compromised channel) or to compute $\boldsymbol{k_B}$ from the information Bob sent to Alice. This can be a very hard problem (discrete logarithm for elliptic curves).

Let's play some more with elliptic curves in pari-gp

- Let's see how long it takes to compute k given P and kP:

```
#
bits=50;
p=nextprime(random([2^(bits-1)+1,2^bits-1]));
E=ellinit([0,0,0,1,1],p);
P=random(E);
o=ellorder(E,P)
k=random([2,o-2])
Q=ellmul(E,P,k);
elllog(E,Q,P)
```

Can we do RSA-like things with elliptic curves?

- No...

```
bits=100;
p=nextprime(random([2^(bits-1)+1,2^bits-1]));
E=ellinit([0,0,0,1,1],p);
P=random(E);
o=ellorder(E,P);
k=0;while(gcd(k,o)!=1,k=random([2,o-2]);); /* public multiplier */
Q=ellmul(E,P,k);
kInv=lift(1/Mod(k,o)); /* private multipler used for decoding   */
R=ellmul(E,Q,kInv)     /* we recover P                          */
```

- But here we do not have an easy to use a hidden secret.
- We would need a point in an elliptic point for which it would be extremely difficult to compute its order.

If you want to know more

- Edwards curves (alternative parameterization of elliptic curves) — paper about them
- "Safe" elliptic curves
- Curve 25519, wikipedia

Secret sharing

Problem:

- n persons what to share a secret.
- **Any** group of t persons can recover the secret.
- Obviously, $n \geqslant 1$ and $1 \leqslant t \leqslant n$.
- On a computer program, the secret will ultimately be an integer.

How to do it:

- A central entity prepares and distributes part of the secret (a secret share) to each person.

Hurdle to overcome:

- Knowing $t - 1$ shares of the secret **must** not give **any** information about the secret.

Secret sharing (how to do it, idea 1, when $n = t$)

- Let the secret be the integer S, and let it have k bits.
- Let the first $n - 1$ shares of the secret, s_1 to s_{n-1}, be random integers with k bits.
- Let the last share of the secret be the exclusive-or of the secret with all the other shares of the secret ($\oplus$ denotes here the bit-wise exclusive-or binary operator):

$$s_n = S \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_{n-1}.$$

- To recover the secret it is only necessary to perform an exclusive-or of all secret shares:

$$S = s_1 \oplus s_2 \oplus \cdots \oplus s_n.$$

- Knowledge of $n - 1$ secret shares does not give any information about the secret.
- It is possible to replace the bit-wise exclusive-or operations by addition and subtractions modulo m. In this case, the first $n - 1$ secret shares are random integers from 0 to $m - 1$, and the last secret share is $(S - s_1 - s_2 - \cdots s_{n-1}) \bmod m$. To recover the secret it is only necessary to add all secret shares (modulo m, of course).

Secret sharing (how to do it, idea 2)

Blakley's secret sharing scheme:

- The secret is a point $\boldsymbol{P}$ in a $\boldsymbol{t}$-dimensional space.
- Each share of the secret is a linear equation (with $\boldsymbol{t}$ unknowns) that has $\boldsymbol{P}$ has one of its solutions.
- Putting together $\boldsymbol{t}$ equations allows us to find $\boldsymbol{P}$.
- It is necessary to ensure that the system of equations has a unique solution for all possible $C_t^n = \frac{n!}{t!(n-t)!}$ possible combinations of $\boldsymbol{t}$ equations chosen from the $\boldsymbol{n}$ equations.
- Each share of the secret is composed by $\boldsymbol{t+1}$ numbers.
- Improved security: the secret is kept **only** in one of the coordinates of the point $\boldsymbol{P}$.
- Modular arithmetic should be used (why?).

Secret sharing (how to do it, idea 3)

Shamir's secret sharing scheme:

- The secret in the independent coefficient a_0 of a polynomial of degree $t-1$,

$$A(x) = \sum_{k=0}^{t-1} a_k x^k.$$

- Each secret share in the pair $\left(x_k, A(x_k)\right)$.
- It is necessary to ensure that distinct values of x_k are used.
- Again, modular arithmetic should be used (why?).
- Each share of the secret is composed by only 2 numbers.

Things to think about:

- Can we do it using square matrices for the a_k coefficients?
- And how about for the a_k coefficients and for the x_k values?

Secret sharing (polynomial interpolation)

Given points (x_k, y_k), for $k = 0, 1, \ldots, n$, with $x_i \neq x_j$ for $i \neq j$, compute the unique polynomial of degree n that passes through these points.

- Newton's interpolation formula:

  $$P_0(x) = y_0,$$

  and, for $k = 1, 2, \ldots, n$,

  $$P_k(x) = P_{k-1}(x) + (y_k - P_{k-1}(x_k)) \frac{(x - x_0) \cdots (x - x_{k-1})}{(x_k - x_0) \cdots (x_k - x_{k-1})}.$$

- Lagrange's interpolation formula:

  $$P_n(x) = \sum_{k=0}^{n} y_k \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i}.$$

If arithmetic modulo p is used we must have $x_i \not\equiv x_j \pmod{p}$ for $i \neq j$. If so, all modular inverses needed by Newton's or Lagrange's intepolation formulas exist.

Quadratic residues

- Let n be a positive integer and let a be an integer such that $\gcd(a, n) = 1$.
- a is said to be a quadratic residue modulo n if and only if there exists a x such that

$$x^2 \equiv a \pmod{n}.$$

- When n is a prime number ($n = p$) there exist three cases:
  1. either a is a multiple of p, or
  2. a is a quadratic residue, or
  3. a is a not a quadratic residue (a quadratic nonresidue).
- The **Legendre** symbol $\left(\frac{a}{p}\right)$ captures this as follows

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{if } p \text{ divides } a, \\ +1, & \text{if } p \text{ does not divide } a \text{ and } a \text{ is a quadratic residue modulo } p, \\ -1, & \text{if } p \text{ does not divide } a \text{ and } a \text{ is a quadratic nonresidue modulo } p. \end{cases}$$

Quadratic residues (Legendre symbol)

- For $p > 2$ the Legendre symbol satisfies the equation

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

(Recall that from Fermat's little theorem we know that $a^{p-1} \equiv 1 \pmod{p}$ when p does not divide a.)

- So, $\left(\frac{a}{p}\right) = \left(\frac{a \bmod p}{p}\right)$.
- In particular, it is possible to prove that

$$\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}, \quad \text{that} \quad \left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}, \quad \text{and that} \quad \left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right).$$

- If q is an odd prime, we also have (this is the famous law of quadratic reciprocity)

$$\left(\frac{q}{p}\right) = (-1)^{\frac{(p-1)(q-1)}{4}} \left(\frac{p}{q}\right).$$

- These properties allow us to easily compute the Legendre symbol for any a and p.

Quadratic residues (Legendre symbol computation)

- Example: Compute $\left(\frac{-14}{73}\right)$.
- $\left(\frac{-14}{73}\right) = \left(\frac{-1}{73}\right)\left(\frac{2}{73}\right)\left(\frac{7}{73}\right)$
- $\left(\frac{-1}{73}\right) = (-1)^{36} = +1$.
- $\left(\frac{2}{73}\right) = (-1)^{666} = +1$.
- $\left(\frac{7}{73}\right) = (-1)^{108}\left(\frac{73}{7}\right) = \left(\frac{3}{7}\right)$.
- $\left(\frac{3}{7}\right) = (-1)^{3}\left(\frac{7}{3}\right) = -\left(\frac{1}{3}\right) = -1$. (Obviously, $\left(\frac{1}{p}\right) = +1$.)
- So, putting it all together, we have $\left(\frac{-14}{73}\right) = -1$
- pari-gp agrees (in pari-gp the Legendre symbol can be computed with the `kronecker` function):

```
kronecker(-14,73)  /* returns -1 */
```

Quadratic residues (Jacobi symbol)

- The **Jacobi** symbol is an extension of the Legendre symbol to the case where the modulus is not a prime number.
- Let $n = p_1^{m_1} p_2^{m_2} \cdots p_k^{m_k}$.
- The Jacobi symbol $\left(\frac{a}{n}\right)$ — yes, it is denoted in exactly the same way as the Legendre symbol — is given by

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{m_1} \left(\frac{a}{p_2}\right)^{m_2} \cdots \left(\frac{a}{p_k}\right)^{m_k}.$$

  (The right-hand side of this formula uses Legendre symbols!)

- Its properties are similar to those of the Legendre symbol, but we also have

$$\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right).$$

- If $\left(\frac{a}{n}\right) = -1$ then a is **not** a quadratic residue modulo nm. **But**, if $\left(\frac{a}{n}\right) = +1$ then a may, or may not, be a quadratic residue modulo nm.

Quadratic residues (counts)

- For a prime p, the number of integers belonging to the set $\{1, 2, \ldots, p-1\}$ that are quadratic residues is exactly $(p-1)/2$.

```
a(n)=local(v,s);v=vector(eulerphi(n));s=0;\
     for(k=1,n,if(gcd(k,n)==1,s=s+1;v[s]=k;));return(v);
qr(n)=local(v);v=a(n); /* number (#) of true quadratic residues */\
     return(length(Set(vector(length(v),k,v[k]^2%n))));
j(n)=local(v,c);v=a(n); /* # of true or fake quadratic residues */\
     return(sum(k=1,length(v),kronecker(v[k],n)==1));
f(n)=return([eulerphi(n),qr(n),j(n)]);
f(101) /* returns [100,50,50] */
f(103) /* returns [102,51,51] */
f(107) /* returns [106,53,53] */
f(109) /* returns [108,54,54] */
```

- How about composite numbers that are the product of two distinct prime numbers?

```
f(13*17) /* returns [192,48,96] --- half are fakes! */
f(11*13) /* returns [120,30,60] --- half are fakes! */
f(11*19) /* returns [180,45,90] --- half are fakes! */
```

Quadratic residues (square roots)

- Let p be a prime number of the form $4k+3$ and let a be a quadratic residue modulo p, i.e., $\left(\frac{a}{p}\right) = +1$.
- Then a has two **square roots**.
- They are given by the formula $r = \pm a^{\frac{p+1}{4}} \bmod p$. This is so because if a is a quadratic residue we must have, by Fermat's little theorem, that $a^{\frac{p-1}{2}} = 1 \bmod p$, and so $a^{\frac{p+1}{2}} = a \bmod p$. But $(p+1)/2$ is an even number so the square roots can be computed easily as stated above.
- If n is the product of two primes p and q of the form $4k+3$ and if a is a quadratic residue modulo n then a will have **four** square roots. They can be easily computed using the Chinese remainder theorem. Two of them will have a Jacobi symbol of $+1$ and two will have a Jacobi symbol of -1.
- Example:

```
p=11; q=19; n=p*q; r=20; a=lift(Mod(r^2,n));
rp=lift(Mod(a,p)^((p+1)/4)); rq=lift(Mod(a,q)^((q+1)/4));
r1=lift(chinese(Mod( rp,p),Mod( rq,q))); /*  20, (r1/p)=+1 */
r2=lift(chinese(Mod( rp,p),Mod(-rq,q))); /*  75, (r2/p)=+1 */
r3=lift(chinese(Mod(-rp,p),Mod( rq,q))); /* 134, (r3/p)=-1 */
r4=lift(chinese(Mod(-rp,p),Mod(-rq,q))); /* 189, (r4/p)=-1 */
```

Quadratic residues (square roots and factorization)

- Let n be the product of two primes.
- Let a be a quadratic residue modulo n.
- Then, it will have four square roots.
- Let x and y be two of then.
- Then $x^2 = y^2 \bmod n$, i.e., $x^2 - y^2 = (x-y)(x+y) = 0 \bmod n$.
- If $y = x$ or $y = -x$, then the above equation gives us nothing.
- Otherwise, we can factor n. Just compute $\gcd(x-y, n)$ and $\gcd(x+y, n)$.
- Example (continuation of the code of the previous slide):

```
p=11; q=19; n=p*q;
r1=20; r2=75; r3=134; r4=189; /* square roots of 191 */
gcd(r1-r2,n);                 /* 11                   */
gcd(r1+r2,n);                 /* 19                   */
```

- pari-gp can only compute square roots when the modulus is prime:

```
sqrt(Mod(5,11))  /* ok (because 5 is a quadratic residue)    */
sqrt(Mod(9,14))  /* error (because the modulus is not prime) */
```

Zero-Knowledge

Introduction (not yet done)

One of two oblivious transfer

- Alice holds two items of information, say m_0 and m_1.
- Bob wants to know one of these two items of information, but does not want Alice to know which one he wants.
- This problem is known as the oblivious transfer problem (in the case, one of two).
- It can be solved in several ways. We will do it here using RSA techniques.
- N is Alice's public RSA modulus and e is the corresponding public exponent; d is the corresponding private decryption exponent.
- At Bob's request, Alice generates two random messages x_0 and x_1 (random numbers smaller than N) and sends then to Bob.
- Bob wants m_b, where $b \in \{0, 1\}$. So, Bob generates a random k and computes and sends to Alice $v = (x_b + k^e) \bmod N$.
- Alice computes $m'_0 = m_0 + (v - x_0)^d \bmod N$ and $m'_1 = m_1 + (v - x_1)^d \bmod N$ and sends both to Bob. Either $(v - x_0)^d \bmod N$ or $(v - x_1)^d \bmod N$ will be equal to k, but Alice has no way of knowing which one is the case.
- Bob computes $m_b = m'_b - k \bmod N$. He can not infer m_{1-b} from m'_{1-b}.

Coin flipping

- Alice and Bob want to flip a coin (by telephone in Manuel Blum's 1981 paper) to decide who wins (in Blum's paper, who gets the car after a divorce).
- Actually, each one flips a coin and if they came out equal (two heads or two tails) Bob wins.
- How can this be done **fairly** and **without cheating** when the two are far apart?
- Using computers: square roots of quadratic residues!

  1. One of the two, say Bob, selects two large random primes p and q of the form $4k+3$ and then computes $n = pq$. He then sends n to Alice.
  2. Alice chooses a random b and sends $a = b^2 \bmod n$ to Bob.
  3. Bob computes the 4 square roots $\pm x$ and $\pm y$ of a, chooses one of then, lets call it r, and sends it to Alice.
  4. Alice checks if $\pm r = b$. If so, then Bob wins. If not, he looses.
  5. Alice proves her claims by disclosing b. (Observe that if Alice does not like the outcome she may simply do not finish the execution of this protocol, but that would be cheating.)

- To flip m coins, do step 2 m times, then step 3 m times and so on. It has to be done in this way because as soon as Alice receives the square roots from Bob she will likely be able to factor n (and so be able to change her choice of the b).

Zero-knowledge proofs of identity (main idea)

- Let us introduce two new protagonists:
  1. **Peggy**, who wishes to prove to Victor that she knows a secret
  2. **Victor**, who wishes to verify that Peggy knows the secret
- The proof will be based on challenge-response pairs and it will be probabilistic in nature.
- The probability that an impersonator is accepted (false proof) decreases as more challenge-response pairs are used.
- One of the first published ways to do it uses (again) the hardness of factoring large integers.
- Again, the underlying problem is computing **square roots** modulo $n = pq$.

Zero-knowledge proofs of identity (Feige-Fiat-Shamir scheme)

Preparatory steps (disclosure of public information):

- Peggy chooses a large number n that is the product of two primes of the form $4k+3$ (such a number is called a Blum number). The interesting thing here is that -1 is not a quadratic residue modulo n but its Jacobi symbol has value $+1$; $x^2 \equiv -1 \pmod{pq}$ implies $x^2 \equiv -1 \pmod{p}$ and $x^2 \equiv -1 \pmod{q}$, so -1 can only be a quadratic residue modulo pq if it is a quadratic residue modulo both p and q, which is not the case here because -1 is not a quadratic residue for primes of the form $4k+3$.
- She also chooses k large random numbers $S_1, S_2, \ldots, S_k$ coprime to n.
- Finally, she also chooses each I_j (randomly and independently) as $\pm S_j^{-2} \bmod n$. The interesting thing here is that no matter which choice was made we always have $\left(\frac{I_j}{n}\right) = +1$, so without computing square roots an external observer cannot determine which choice was made. The S_j are witnesses of the quadratic character of the I_j.
- She publishes n and the $I = I_1, I_2, \ldots, I_k$ (but keeps $S = S_1, S_2, \ldots, S_k$ secret).

Instead of publishing n herself, Peggy could have used any Blum integer computed by a trusted entity (the factors of n are not used anywhere in this scheme.)

Zero-knowledge proofs of identity (Feige-Fiat-Shamir scheme)

To generate and verify a proof of identity, Peggy and Victor execute the following T times (the higher T is the harder it will be to fake the proof of identity):

- Peggy chooses a random R and sends to Victor $X = \pm R^2 \mod n$. Here she also chooses the sign, either $+$ or $-$ randomly, so X is, or isn't a quadratic residue. (Remember, zero knowledge leaked!)
- **[The challenge]** Victor send to Peggy the random vector of bits $E = E_1, E_2, \ldots, E_k$; each E_j is either 0 or 1.
- **[The reply]** Peggy computes and sends to Victor $Y = \pm R \prod_{E_j=1} S_j \mod n$; here, again, she chooses the sign in a random way.
- **[The verification]** Victor checks if $X = \pm Y^2 \prod_{E_j=1} I_j \mod n$, and rejects immediately the proof if this is not so.
- Anyone trying to impersonate Peggy (Eve?) could try to guess the E_j — let the guesses be E'_j — them precompute the next round of the protocol by selecting a random Y and by presenting $X = \pm Y^2 \prod_{E'_j=1} I_j$ when so requested. The probability of success of this cheating attempt is 2^{-k} per round (so, 2^{-kT} overall).

Bibliography (work in progress)

1. Eric Bach and Jeffrey Shallit, **Algorithmic Number Theory, Volume 1, Efficient Algorithms**, MIT Press, 1996.

2. Richard Crandall and Carl Pomerance, **Prime Numbers. A Computational Perspective**, second edition, Springer-Verlag, 2005.

3. Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno, **Cryptography Engineering. Design Principles and Practical Applications**, Wiley, 2010.

4. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, **Handbook of Applied Cryptography**, fifth printing, CRC Press, 2001.

5. Richard A. Mollin, **Advanced Number Theory with Applications**, CRC Press, 2009.

6. Richard A. Mollin, **Codes. The Guide to Secrecy from Ancient to Modern Times**, Chapman & Hall/CRC, 2005.

7. Bruce Schneier, **Applied Cryptography. Protocols, Algorithms, and Source Code in C**, second edition, Wiley, 1996.