* Cubics to hide messages

Master Course, Spring term 2025

Fernando Chamizo https://matematicas.uam.es/~fernando.chamizo/

Contents. Public key cryptography. The discrete logarithm problem. Diffie-Hellman key exchange. ElGamal cryptosystem. Elliptic curve cryptography.

5.1 Trapdoors, public keys and logarithms

Recall that the encryption and decryption functions depend on the key. Roughly speaking, in *public key cryptography* the encrypter is provided with a pre-processed version of the key that is enough for the encryption process but, in practice, it is impossible to recover the key from it. In this way, a person can receive ciphered messages from general senders without revealing the key.

In mathematical terms, the key can be considered a pair $(k_{\rm pri}, k_{\rm pub})$, composed by a *private key*, $k_{\rm pri}$, and a *public key*, $k_{\rm pub}$, with $k_{\rm pub} = f(k_{\rm pri})$ for some complicated one-to-one function f. The encryption algorithm only uses $k_{\rm pub}$ that can be made public while the decryption algorithm employs $k_{\rm pri}$. It is said to be an *asymmetric cipher* because not all the persons involved in the communication have access to the same information [5, 1.7.6]. Everybody can know how to decrypt using $k_{\rm pri}$, the point is that $k_{\rm pri} = f^{-1}(k_{\rm pub})$ is very hard to compute in practice and $k_{\rm pub}$ turns out to be useless to decrypt ciphertexts. It is said that $k_{\rm pri}$ is a *trapdoor* to invert the encryption function getting the decryption function.

In cryptography the most conspicuous example of a function difficult to invert in practice is the exponentiation in finite fields or some finite groups. Let us examine the first case.

The finite field $\mathbb{F}_p$ composed by the classes modulo p (prime) has a cyclic multiplicative group $\mathbb{F}_p^*$ [15, 2.5]. In other words, there exists $g \in \mathbb{F}_p^*$, called a *primitive root* modulo p when considered in $\mathbb{Z}$, such that $\langle g\rangle = \mathbb{F}_p^*$. The following statement should be fairly trivial:

Lemma 5.1. *If g is a primitive root modulo p then $\phi : \mathbb{Z}/(p-1)\mathbb{Z} \longrightarrow \mathbb{F}_p^*$ given by $\phi(n) = g^n$ defines an isomorphism.*

Note that it implies that $g^x \equiv m \pmod p$ has exactly a solution x modulo $p-1$ when $p \nmid m$ and it includes *Fermat's little theorem* $a^{p-1} \equiv 1 \pmod p$ for $p \nmid a$ (just write $a \equiv g^n$).

With a standard computer, it takes almost no time to compute a^b modulo p for a, b and p of hundreds of digits. For instance, under sagemath

```
# Mersenne prime with more than 180 digits
p = 2^607-1
a, b  = 10^200, 10^300
print( Mod(a,p)^b )
```

took few miliseconds to compute the answer.

We claim that ϕ is "easy" to compute (for an efficient algorithm to compute powers modulo n see [15, §2.3.2] or [5, §1.3.2]). The important point is that there are not good algorithms to "take logarithms" modulo p getting ϕ^{-1} i.e., to solve the equation $g^x \equiv m \pmod{p}$. This is called the *discrete logarithm problem* by obvious reasons. The value of x is the *discrete logarithm* of m to base g.

For instance,

```
k = 34
q = next_prime(10^(k+1))
a, b = 2, 10^k
print ( log( Mod(a,q)^b, Mod(2,q) ) )
```

involving the prime $q = 10^{35}+69$ took around 40 seconds to give the right answer, 10^{34}, solving $2^x \equiv 2^{10^{34}} \pmod{q}$. With today knowledge, a computer could not solve a problem like this involving hundreds of digits.

If p has some tenths of digits, it is in general impossible for a computer to solve the discrete logarithm problem by brute force trying g^0, g^1, g^2, g^3, ... until a power gives the expected value m because it could require a number of power computations comparable to p. There is an ingenious observation that allows to reduce this number to something comparable to $\sqrt{p}$, which is a great gain, but still unfeasible for hundreds of digits (just for comparison, it is believed that the number of protons in the observable universe is comparable to 10^{80}). The ingenious observation is utterly simple:

Lemma 5.2 (Babystep-Giantstep algorithm)**.** *Let g and p as before, $p \nmid m$ and $N = 1 + \lfloor\sqrt{p-1}\rfloor$. Then $\{g^k \pmod{p}\}_{k=0}^{N-1} \cap \{mg^{-kN} \pmod{p}\}_{k=0}^{N-1} \neq \emptyset$, say $g^{k_0} \equiv mg^{-k_1 N} \pmod{p}$, and hence $x = k_0 + k_1 N$ solves $g^x \equiv m \pmod{p}$.*

Proof. By Lemma 5.1 there is a solution of $g^x \equiv m$ and we can restrict it to the range $0 \leq x \leq p-1 < N^2$. Dividing x by N they are obtained a quotient k_1 and a remainder k_0 less than N and $g^{k_0+k_1 N} \equiv m$ is the same as $g^{k_0} \equiv mg^{-k_1 N} \pmod{p}$. □

The point is that the geometric progressions composing the two sets in Lemma 5.2 take N multiplications each to be constructed (for the first one multiplies by the *babystep* g and for the second by the *giantstep* g^{-N}) and computing an element of the intersection with sorting and searching algorithms takes a logarithmic number of steps.

There are other clever tricks to reduce the effort to attack the discrete logarithm problem in any finite field [1]. Anyway, nothing spectacular has been proved or, at least, published in the last decades suggesting that the problem is approachable for hundreds of digits.

5.2 Exchanging keys and messages

In the making epoch paper [3] it was introduced the nowadays called *Diffie-Hellman key exchange*. It solves a problem in cryptography that seems impossible at first glance: To share safely a secret key through an insecure channel.

Say that *Alice* and *Bob*, the standard fictional main characters in cryptography (aka A and B), agree on a large prime p and a primitive root modulo p (in fact, using a primitive

root is not strictly necessary). This information can be even made public. If Alice sends $g^a \pmod p$ to Bob and Bob sends $g^b \pmod p$ to Alice where a and b are secret (not shared) large random numbers, then both of them can compute $g^{ab} = (g^a)^b = (g^b)^a$ modulo p and take it as a shared secret key. An attacker, named *Eve* in the literature (aka E for eavesdropper), that has tapped the communication channel can know g^a and g^b modulo p but to obtain the secret key, Eve would have to solve the *Diffie-Hellman problem*: To compute $g^{ab} \pmod p$ knowing g^a and g^b modulo p.

Solving the discrete logarithm problem allows to solve the Diffie-Hellman problem. Although the converse is not known, it is considered that they are the same animal. There are not practical general algorithms to solve the Diffie-Hellman problem when p is huge and cryptography takes advantage of it.

The Diffie-Hellman key exchange is a protocol to decide about a key. A variant of the idea closely related to the discrete logarithm problem leads to *ElGamal cryptosystem* [4]. The plaintexts and the cipher texts are $\mathcal{P} = \mathbb{F}_p^*$ and $\mathcal{C} = \mathbb{F}_p^* \times \mathbb{F}_p^*$ with p prime, the key is $k = (k_{\text{pri}}, k_{\text{pub}})$ with $k_{\text{pub}} \in \mathbb{Z}/(p-1)\mathbb{Z}$ and $k_{\text{pub}} = g^{k_{\text{pri}}} \in \mathbb{F}_p^*$ where, as before, $\langle g \rangle = \mathbb{F}_p^*$. The encryption and decryption functions are

$$e(k, m) = \big(g^r, m k_{\text{pub}}^r\big) \qquad \text{and} \qquad d\big(k, (c_1, c_2)\big) = c_2 c_1^{-k_{\text{pri}}}$$

where r is a large random number that can be changed in each encryption for extra security (it is called an *ephemeral key*). The important point is that the encryption function only involves the k_{pub} part of the key. Thanks to the discrete logarithm problem, Eve cannot recover r from the first coordinate of $e(k, m)$ and hence she cannot compute k_{pub}^r to get m from the second coordinate. If Alice keeps k_{pri} secret she will be able to receive messages from Bob and from any other person knowing k_{pub} and none of them will be able to decrypt messages that they have not encrypted without solving the discrete logarithm problem.

A possible sagemath implementation of the encryption and decryption functions is

```
# ElGamal
# pub_key = public key
# pri_key = private key
# g = generator or high order element
# p = prime
# message = number < p
# ciphertext (m_1,m2) = pair of numbers < p

def elgamal_encrypt(pub_key,g,p,message):
    k = randint(1,p-2)
    return (Mod(g,p)^k, message*Mod(pub_key^k,p) )

def elgamal_decrypt(pri_key,g,p,(m1,m2)):
    return Mod(m2,p)*Mod(m1,p)^(-pri_key)
```

Probably the most famous public key cryptosystem is the *RSA cryptosystem* [11]. We talk here briefly about it without entering into the details (the full third chapter of [5] is devoted to it). It is cryptosystem based on factorization. The private key is composed of two large primes $k_{\text{pri}} = (p, q)$ and the public key is $k_{\text{pub}} = (N, e)$ with $N = pq$ and e an encrypting number that satisfies $\gcd\big(e, (p-1)(q-1)\big) = 1$. The plaintexts and ciphertexts are $(\mathbb{Z}/N\mathbb{Z})^*$ (this dependence on the key can be avoided imposing some prior ranges for p and q). The

encryption and decryption functions are

$$e(k,m) = m^e \pmod N \qquad \text{and} \qquad d(k,c) = c^{e_*} \pmod N$$

where e_* is the inverse of e modulo $(p-1)(q-1)$. Checking that they are valid functions requires to note $m^{(p-1)(q-1)} \equiv 1 \pmod N$ by *Euler's congruence* $a^{\varphi(n)} \equiv 1 \pmod n$ for a and n coprime [12, Ch. 6]. Then

$$d\big(k, e(k,m)\big) \equiv (m^e)^{e_*} = m^1 + k(p-1)(q-1) \equiv m \pmod N.$$

If p and q are given, computing e_* reduces to an application of Euclid's algorithm, that requires a number of steps at most comparable to the number of digits of N [12]. The expected security of the algorithm lies in the absence of good methods to factorize general numbers of, say, hundreds of digits. It is fair to add that, with present knowledge, one cannot assert mathematically that solving $x^e \equiv c \pmod N$ necessarily requires to factorize N. We can only say that factorization is the only known method to proceed in practical cases. There is a factorization algorithm, *Shor's algorithm* [14], [10] that running on a good enough quantum computer could crack RSA with current ranges. Although there is a lot of fuss on it to promote quantum computation, it is so far a quite theoretical result. In practice, the largest number factorized in this way by a quantum computer seems to have two digits! Recent advances in quantum computing have not changed this benchmark.

5.3 The elliptic siblings

There exists an *elliptic curve cryptography* [8], [7] and part of it replaces the multiplicative group of $\mathbb{F}_p$ by the group $E(\mathbb{F}_p)$ composed by the points of an elliptic curve E over $\mathbb{F}_p$. The motivation is that in $E(\mathbb{F}_p)$ the analog of the discrete logarithm problem is less approachable con with our present knowledge (although in some special cases it can be translated to the problem in finite field [5, §6.9.1] or even to an easier one [13]).

Recall that the discrete logarithm problem consists in solving $g^x = h$ in $\mathbb{F}_p^*$ for given g and h (and p). The *elliptic curve discrete logarithm problem* is the analogue changing the multiplicative group operation by the group law for $E(\mathbb{F}_q)$ with $\mathbb{F}_q$ a finite field (we only consider here the case $q = p$ prime). It consists in finding $x \in \mathbb{Z}$ such that $xG = H$ where G and H are points on a given elliptic curve E over $\mathbb{F}_q$. We say that x is the *discrete logarithm* of H to the base G.

In sagemath it can be solved with `G.dicrete_log(H)`. For instance

```
E = EllipticCurve(GF(103), [2,1])
G = E([0,1])
H = 20*G
print ( G.discrete_log(H) )
```

prints `20`. Here `[2,1]` means that the elliptic curve is $y^2 = x^3 + 2x + 1$. If we replace `20` by `110` the result is `4` because the order of G is 106. By the way, the latter value is obtained with `additive_order(G)`.

In general, nothing better that something comparable to $\sqrt{p}$ steps (which corresponds to Lemma 5.2) is known to solve the elliptic curve discrete logarithm problem [5, §6.3.2]. This

means that using p with a hundred digits (or even much less) is safe. In the previous listing changing `GF(next_prime(103))` by `GF( next_prime(10^20))` could be too much for sagemath running on a usual computer.

Of course, in applications one looks for G having large order. Two results of the theory of elliptic curves are relevant here. One is that $E(\mathbb{F}_p)$ can be a cyclic group or the product of two cyclic groups (this comes from considering the $|E(\mathbb{F}_p)|$-division points [2]) and the order of $E(\mathbb{F}_p)$ is always close to p, namely, it differs from $p+1$ in less than $2\sqrt{p}$ (this is Hasse's theorem [6, X.3]). Both results assure the existence of $G \in E(\mathbb{F}_p)$ with large order if p is large.

In sagemath, `E.abelian_group()` gives the structure of the abelian group of an elliptic curve `E`. On the other hand, `E.gens()` gives a list with the generators in such a way that the first one has maximal order.

```
E = EllipticCurve(GF(47), [1,1])
print (E.abelian_group())
print (E.gens())
print ('Element of maximal order =',E.gens()[0])
```

A possible output for this code is:

```
(Multiplicative Abelian Group isomorphic to C30 x C2, ((44 : 21 : 1),(35 : 0 : 1)))
((44 : 21 : 1), (35 : 0 : 1))
Element of maximal order = (44 : 21 : 1)
```

The format of `E.abelian_group()` can vary from a version of sagemath to another. The previous output means that $P = (44, 21)$ and $Q = (35, 0)$ are points of order 30 and 2, respectively and any point on E can be written as $mP + nQ$ with $m, n \in \mathbb{Z}$.

The Diffie-Hellman key exchange and the ElGamal cryptosystem can be adapted to the elliptic curves setting changing $\mathbb{F}_p^*$ by $E(\mathbb{F}_p)$.

For Diffie-Hellman key exchange, Alice sends aG, Bob sends bG, with a and b large integers, and both can compute $abG = a(bG) = b(aG)$.

For ElGamal cryptosystem, a point $G \in E(\mathbb{F}_p)$ of large order and E itself (including the field $\mathbb{F}_p$ in which it is defined) are public information. The key $k = (K_{\text{pub}}, k_{\text{pri}})$ is composed by the private key k_{pri} given by a positive integer less than the order of G and the public key $K_{\text{pub}} = k_{\text{pri}}G$. The hardness of elliptic discrete logarithm problem assures that it is difficult to recover K_{pub} from k_{pri}.

The sets of plaintext messages and ciphertexts are the group $E(\mathbb{F}_p)$ of points over $\mathbb{F}_p$. The encryption and decryption functions are

$$\begin{aligned} e(k, M) &= (rG, M + rK_{\text{pub}}) \qquad r = \text{ random number}, \\ d\big(k, (C_1, C_2)\big) &= C_2 - k_{\text{pri}}C_1. \end{aligned}$$

A technical problem is how to encode characters into points of an elliptic curve. This is solved in several ways in [8]. There is also a variation of the cryptosystem sometimes called *MV-ElGamal* (MV stands for Menezes and Vanstone) that avoids this technical problem in a simple way [5, p. 365]. In this version a message is divided into two blocks m_1 and m_2 modulo p i.e., $\mathcal{P} = \mathbb{F}_p \times \mathbb{F}_p$ is the set of plaintext messages (and the encoding is very easy).

The encryption function is given by

$$e(k, M) = \big(rG, (c_1, c_2)\big) \in E \times \mathbb{F}_p \times \mathbb{F}_p$$

where $c_1 \equiv xm_1 \pmod p$, $c_2 \equiv ym_2 \pmod p$, with $(x, y) = rK_{\text{pub}}$. We assume $x, y \in \mathbb{F}_p^*$, otherwise we choose another random r. The corresponding decryption function is

$$d\big(k, C_0, (c_1, c_2)\big) = (c_1 x^{-1}, c_2 y^{-1}) \qquad \text{where } (x, y) = k_{\text{pri}} C_0$$

because for $C_0 = rG$ it holds $k_{\text{pri}} C_0 = r k_{\text{pri}} G = rK_{\text{pub}}$.

For instance, if we choose

```
#
# Choose the elliptic curve modulo p = large prime
# and G a point of high order
#
p = next_prime(10^10)
E = EllipticCurve( GF( p ), [2011,1])
G = E([0,1])
print ( G.additive_order() )
```

The output is `3333330247`, then the order G is quite large.

The functions e and d introduced above can be coded as:

```
#
# Encryption and decryption functions
#
def encrypt_mv_eg(Kpub,m1,m2):
    x,y = 0,0
    while( (x==0) or (y==0) ):
        r = randint( 1, p-1 )
        x = (r*Kpub)[0]
        y = (r*Kpub)[1]
    return r*G, m1*x, m2*y

def decrypt_mv_eg(kpri,enc):
    x = (kpri*enc[0])[0]
    y = (kpri*enc[0])[1]
    return enc[1]*x^-1, enc[2]*y^-1
```

If it is a valid cryptosystem then $d\big(k, e(k, M)\big) = M$

```
#
# Example
#
private_key = 12345
public_key = private_key*G
decrypt_mv_eg(private_key, encrypt_mv_eg(public_key,10101,33333))
```

We recover the original message `(10101, 33333)`.

A final comment is that elliptic curves can be used to factorize large numbers employing a very clever, not specially deep, algorithm [9] (see [2, §20] for a quick explanation). In this sense, one can devise elliptic curve attacks to RSA.

Exercises

Exercise 1. Why are there $\varphi(p-1)$ primitive roots modulo p with φ Euler's totient function?

Exercise 2. Let G be a finite abelian group. Show that G is cyclic if and only if the least common multiple of the order of all elements N equals $|G|$. Using this fact and that $x^N - 1 = 0$ has at most N solutions in $\mathbb{F}_p$, deduce that $\mathbb{F}_p^*$ is cyclic.

Exercise 3. Implement in sagemath a version of the Babystep-Giantstep algorithm to decide if for given $a, m \in \mathbb{Z}$ and p prime $a^x \equiv m \pmod{p}$ has solutions computing one of them in the affirmative.

Exercise 4. Consider ElGamal cryptosystem with $p = 2^{31} - 1$ and $g = 7$. Use sagemath to decrypt $(297629860, 1094924871)$ knowing that the public key is 1659750829.

Exercise 5. Check that the sum of two points $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2) \in E - \{O\}$ with different abscissae in the elliptic curve $y^2 = x^3 + ax + b$ is given by

$$(m^2 - x_1 - x_2, -mx_3 - b) \qquad \text{with} \quad m = \frac{y_2 - y_1}{x_2 - x_1}, \quad b = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1}.$$

Exercise 6. Show that the formula in the previous exercise works when $P_1 = P_2$ taking $m = \frac{3x_1^2 + a}{2y_1}$, $b = \frac{-x_1^3 + ax + 2b}{2y_1}$.

Exercise 7. Consider $E(\mathbb{F}_5)$ with $E : y^2 = x^3 + 4x + 3$ and compute by hand $2024^{2025}P$ for $P = (2, 2)$.

Exercise 8. Establish a formula for the number of elliptic curves $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ with $p > 3$.

References

[1] L. M. Adleman and J. DeMarrais. A subexponential algorithm for discrete logarithms over all finite fields. *Math. Comp.*, 61(203):1–15, 1993.

[2] J. W. S. Cassels. *Lectures on elliptic curves*, volume 24 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 1991.

[3] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22(6):644–654, 1976.

[4] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31(4):469–472, 1985.

[5] J. Hoffstein, J. Pipher, and J. H. Silverman. *An introduction to mathematical cryptography*. Undergraduate Texts in Mathematics. Springer, New York, 2008.

[6] A. W. Knapp. *Elliptic curves*, volume 40 of *Mathematical Notes*. Princeton University Press, Princeton, NJ, 1992.

[7] A. H. Koblitz, N. Koblitz, and A. Menezes. Elliptic curve cryptography: the serpentine course of a paradigm shift. *J. Number Theory*, 131(5):781–814, 2011.

[8] N. Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177):203–209, 1987.

[9] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Ann. of Math. (2)*, 126(3):649–673, 1987.

[10] M. Nakahara and T. Ohmi. *Quantum computing*. CRC Press, Boca Raton, FL, 2008. From linear algebra to physical realizations.

[11] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21(2):120–126, 1978.

[12] K. H. Rosen. *Elementary number theory and its applications*. Addison-Wesley Publishing Company, Advanced Book Program, Reading, MA, fourth edition, 2000.

[13] I. A. Semaev. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. *Math. Comp.*, 67(221):353–356, 1998.

[14] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[15] W. Stein. *Elementary number theory: primes, congruences, and secrets*. Undergraduate Texts in Mathematics. Springer, New York, 2009. A computational approach.