Elliptic Curves

Guilhem Castagnos

September – December 2017

last updated: September 1, 2017

Contents

Chapitre I

The Discrete Logarithm Problem

1. Bibliography

- In French: *Courbes elliptiques - Une présentation élémentaire pour la cryptographie*, Philippe Guillot, Éditions Hermes-Lavoisier, (2010)

- *Elliptic Curves and Their Applications to Cryptography — An Introduction*, Andreas Enge, Kluwer Academic Publishers (1999), *Bilinear pairings on elliptic curves*, Andreas Enge, L'Enseignement Mathématique, `https://hal.inria.fr/hal-00767404/` (2015)

- *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Henri Cohen, Gerhard Frey, Chapman & Hall/CRC (2006)

- *The arithmetic of Elliptic Curves*, Joseph H. Silverman, Springer, 2nd Edition (2009), in particular chapter XI, *Algorithmic Aspects of Elliptic Curves*

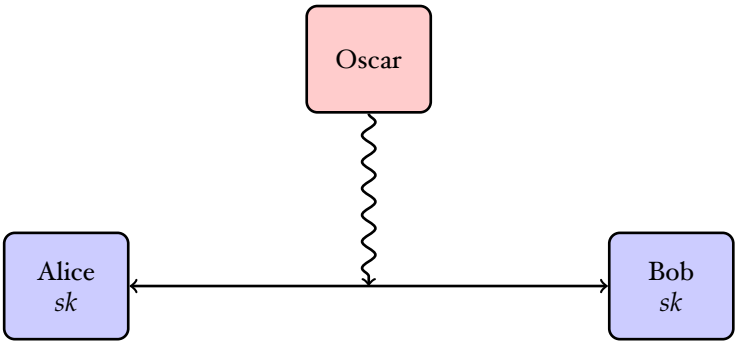
- *Mathematics of Public Key Cryptography*, Steven Galbraith, Cambridge University Press (2012), in particular Part II *Algebraic Groups* and Part V *Cryptography Related to Discrete Logarithms*

2. Asymmetric and Symmetric Cryptography

In cryptography, Elliptic Curves are mainly used for asymmetric cryptography.

Differences between Symmetric and Asymmetric Cryptography: For example, encryption Schemes : to protect confidentiality between Alice and Bob.

Symmetric Encryption Scheme



$$\mathsf{Encrypt}_{sk}(m_A) = c_A \longrightarrow \mathsf{Decrypt}_{sk}(c_A) = m_A$$

$$\mathsf{Encrypt}_{sk}(m_B) = c_B \longleftarrow \mathsf{Decrypt}_{sk}(c_B) = m_B$$

m_A, m_B: plaintexts (*messages clairs*), c_A, c_B: ciphertexts (*messages chiffrés*), Encrypt : encryption scheme (*Algorithme de chiffrement*), Decrypt : decryption scheme (*Algorithme de déchiffrement*), sk: secret or private key

Asymmetric Encryption Scheme

$$\mathsf{Encrypt}_{pk}(m) = c \longrightarrow \mathsf{Decrypt}_{sk}(c) = m$$

pk: public key, sk: private key

Symmetric vs Asymmetric

Symmetric cryptography: security is based on the size of the key (only attack must be exhaustive search: k bits key, 2^k operations), Fast, Encryption (e.g. AES), MAC (for integrity and authentication), Hash Functions (for integrity, e.g. the SHA family),...

Asymmetric cryptography: security is based on the difficulty of on algorithmic problem. Key size is chosen in order to make this problem intractable. Slower than symmetric cryptography, Key exchange (e.g. Diffie-Hellman), Encryption (e.g. RSA), Digital Signature (e.g. DSA, for integrity, authentication and non repudiation).

Security of cryptosystems based on elliptic curves mostly rely on algorithmic problems related to the discrete logarithm problem.

3. The Discrete Logarithm Problem

3.1. Definitions

We consider a cyclic group of order n, $(G, \times)$ generated by $g \in G$, *i.e.*, $\langle g \rangle = \{g, g^2, g^3, \dots, g^n = 1\} = G$. In general, n will be a prime number (cf. Pohlig–Hellman Algorithm).

Examples

For example a subgroup of $(\mathbf{Z}/p\mathbf{Z})^\times$ with p prime. $(\mathbf{Z}/p\mathbf{Z})^\times$ is cyclic of order $p-1$. If n divides $p-1$, there exists $g \in (\mathbf{Z}/p\mathbf{Z})^\times$ of order n. More generally, a subgroup of a finite field $\mathbf{F}_q$ ($q = p^d$) as $(\mathbf{F}_q)^\times$ is cyclic of order $q-1$. The group of points of an Elliptic Curve. Order is by Hasse: $|\mathrm{E}(\mathbf{F}_q) - (q+1)| \leqslant 2\sqrt{q}$, then we consider a subgroup of prime order n. In this case, we will use additive notations: $(\mathrm{G}, +)$ with $\mathrm{G} = \{\mathrm{P}, 2\mathrm{P}, 3\mathrm{P}, \dots, n\mathrm{P} = \mathrm{O}\}$.

The Discrete Logarithm Problem

Let $h \in g$. We know that $h = g^x$ for some x. Finding x is solving the Discrete Logarithm Problem. We will denote $x = log_g(h)$. We can define x modulo n: Let us consider the group morphism $exp_g : (\mathbf{Z}/n\mathbf{Z}, +) \to (\mathrm{G}, \times)$, $a \mapsto g^a$. It is well-defined: if $a = b + kn$, $exp_g(a) = g^a = g^b = exp_g(b)$. It is a morphism : $exp_g(a+b) = exp_g(a) \times exp_g(b)$. It is an isomorphism (a bijective morphism): same number of elements and injective: $exp_g(a) = 1$, $n \mid a$, so $a = 0$. The discrete logarithm is the inverse function, so it is a morphism from $(\mathrm{G}, \times)$ to $(\mathbf{Z}/n\mathbf{Z}, +)$.

Properties

If g and g' are two generators then $g' = g^a$ with a invertible modulo n: indeed, there exits also b such that $g = g'^b = g^{ab}$, so $ab \equiv 1 \mod n$. Conversely, if $h = g^a$ with a invertible then h is a generator: if $h^b = 1$ then $g^{ab} = 1$ so $ab \equiv 0$ and $b \equiv 0$.

Change of generator: If g, g' are two generators, $h = g^a$ et $h = g'^b$, *i.e.*, $log_g(h) = a$ and $log_{g'}(h) = b$. We denote $c = log_g g'$. Then $g' = g^c$ so $h = g^{bc} = g^a$ and $a \equiv bc$. As a result, $log_g(g) = log_{g'}(h) \times log_g g'$, or $log_{g'}(h) \equiv log_g(g)\, log_g g'^{-1}$ where this last term is invertible because g' is a generator.

As a result, if we know how to compute discrete logarithms in basis g then we can do it in basis g'. The difficulty of computing discrete logarithms only depends on the group and not on the generator chosen.

Remark that the discrete logarithm problem can be easy: For example in $\mathrm{G} = (\mathbf{Z}/n\mathbf{Z}, +)$. g is a generator if and only if $gcd(g, n) = 1$. Indeed, we know that 1 is a generator as $\mathrm{G} = \{1, 2, 3, \dots, n = 0\} = \{1, 2 \times 1, 3 \times 1, n \times 1 = 0\}$. From what we saw, the others generators are the $a \times 1$ with a invertible modulo n. Then $(\mathbf{Z}/n\mathbf{Z}, +) = \mathrm{G} = g, 2g, 3g, \dots ng = 0$. If $h \in \mathbf{Z}/n\mathbf{Z}$, $h \equiv xg$, $x \equiv hg^{-1}$ computable in polynomial time with a simple extended Euclidean algorithm.

3.2. Algorithms for Computing Discrete Logarithms

First we see some generic algorithms that work for all cyclic group G of order n: the naive algorithm, The Baby Step Giant Step method , the ρ method of Pollard, and the Pohlig–Hellman algorithm that reduce computing discrete logarithm in a group of order n to computing logarithm in subgroups of order p where p is prime and p divides n. By a theorem of Shoup (97), an algorithm that solve the discrete logarithm in G must do at least $\mathcal{O}(\sqrt{n})$ operations in G.

The Naive Algorithm

$h = g^x$, x?

Compute $g, g^2, g^3, \dots$. Complexity, $\mathcal{O}(n)$ multiplications in G

With memory: We pre-compute all the (g^i, i) and store them in a list, sorted we respect to the g^i (by using a binary representation of the elements of G for instance). Complexity: $\mathcal{O}(n)$ multiplications in G, $\mathcal{O}(n)$ elements of G in memory, sorting algorithm complexity : $\mathcal{O}(n\, log(n))$ Then to compute x s.t. $h = g^x$, we look for h in the list: complexity: $\mathcal{O}(log(n))$

Baby Step/Giant Step

Usually credited to Shanks 1971. Time-memory trade-off.

$h = g^x$ x?

Let $m = \lceil n \rceil$ and $x = i + mj$ with $0 \leqslant i, j < m$. We have $h = g^x = (g^m)^j g^i$. So $h(g^{-1})^i = (g^m)^j$.

Pre-computations: a list of $((g^m)^j, j)$ with $j < m$ sorted with respect to the first coordinate: $\mathcal{O}(\sqrt{n})$ memory, and $\mathcal{O}(\sqrt{n}\,log(n))$ for the sorting algorithm.

Then we compute $h, hg^{-1}, h(g^{-1})^2, \ldots$ and look for this element in the list. Worst case: $\mathcal{O}(\sqrt{n})$ multiplications and complexity for $\mathcal{O}(\sqrt{n}\,log(n))$ the searches.

Pollard ρ

Pollard 1978

Same time complexity but quasi no memory. Probabilistic Algorithm: Sometimes no result is found.

We look for integers modulo n s.t. $g^i h^j = g^{i'} h^{j'}$. Then $g^{i-i'} = h^{j'-j}$. So $i - i' \equiv x(j' - j)$ and if $(j' - j)$ is invertible modulo n, we find x.

To find these integers, one can store all the tuples $(g^i h^j, i, j)$ sorted with respect to the first coordinates, with random i and j. If one finds two times the same element of G (a collision), the discrete logarithm can be extracted as above.

By the birthday paradox with a good probability, there will be a collision for around $\sqrt{n}$ random choices. We draw uniformly k elements in a set of n elements. Let $p(k)$ the probability that there is a collision. The probability that all the elements are different is

$$1 - p(k) = \left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right) \ldots \left(1 - \frac{k-1}{n}\right) = \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right)$$

As $1 - x \leqslant e^{-x}$ for all $x \in \mathbf{R}$, $1 - p(k) \leqslant \prod_{i=1}^{k-1} e^{-i/n} = e^{-k(k-1)/2n} \leqslant e^{-(k-1)^2/2n}$. As a result, there is a collision with probability at least $f(k) := 1 - e^{-(k-1)^2/2n}$. For $k_0 = 1 + \sqrt{-2n\,log(1 - \mathrm{A})}$, one has $\mathrm{A} = f(k_0)$. So for $k \geqslant k_0$, one has $p(k) \geqslant f(k) \geqslant f(k_0) = \mathrm{A}$. As a result, for the probability to be at least 1/2, one must draw $1 + \sqrt{2n\,log\,2} \approx 1.177\sqrt{n}$ elements (for a probability greater than 0.99, this gives $3.03\sqrt{n}$ elements). For the classical birthday problem, $n = 365$, and we have a probability 1/2 of having two people with the same birthday for a set of $1 + \sqrt{2 \times 365\,log\,2} \approx 23.49$ people (In fact 23 is sufficient).

More generally, one can show that the expected number of elements to be drawn until a collision is found is less than $\sqrt{\pi n/2} + 2 \approx 1.253\sqrt{n}$.

As a consequence, this method gives a priori no gain with respect to Baby Step/Giant Step which gives a deterministic method with the same memory and a slightly better time complexity.

To get rid of the need of storage, one can make an iteration of a function $f : \mathrm{G} \to \mathrm{G}$ that « looks like » a random function. That function must satisfy the following property: given $\mathrm{X} \in \mathrm{G}$ and i, j s.t. $\mathrm{X} = g^i h^j$, on can efficiently compute $f(\mathrm{X})$ and i', j' s.t. $f(\mathrm{X}) = g^{i'} h^{j'}$.

Such a function can be obtained as follows. First, one makes a partition of G in sets $\mathrm{S}_1, \ldots, \mathrm{S}_n$. Then f is defined on each S_k with the previous property on the power of g and h. For example, the initial proposal of Pollard consists in partitioning G with three sets $\mathrm{S}_0, \mathrm{S}_1, \mathrm{S}_2$, and setting $f(\mathrm{X}) = \mathrm{X}^2$ if $\mathrm{X} \in \mathrm{S}_0$, $f(\mathrm{X}) = h\mathrm{X}$ if $\mathrm{X} \in \mathrm{S}_1$ and $f(\mathrm{X}) = g\mathrm{X}$ if $\mathrm{X} \in \mathrm{S}_2$. For example, in the case S_0, if $\mathrm{X} = g^i h^j$, than $f(\mathrm{X}) = \mathrm{X}^2 = g^{2i} h^{2j}$. In practice, a larger number of sets (10 to 20) gives better results.

With that function, we start with a random $X_0 \in G$, then we define $X_m = f(X_{m-1})$ pour $m > 0$ and we search for a collision. The graph formed by the values in the sequence gives the ρ shape that gives its name to the method. To find the collision, a cycle-finding algorithm such as the Floyd method is used. That method, also known as the method of «the tortoise and the hare algorithm » is as follows. One iteratively compute the position of the tortoise and the hare: $(X_m, X_{2m}) = (f(X_{m-1}), f \circ f(X_{2(m-1)}))$. For that only the two previous positions are needed, so a few memory is needed. Then we stop when the tortoise and the hare are in the same position, where we obtain the collision.

What is the complexity? Let ℓ be the index of the first element of the cycle and c the period, *i.e*, the length of the cycle. As a result, $X_0, \dots, X_{\ell-1}, X_\ell, \dots, X_{\ell+c-1}$ are all distinct and $X_{m+c} = X_m$ for $m \geqslant \ell$. With the Floyd method, we find the smallest u s.t. $X_u = X_{2u}$ where $u \geqslant \ell$ et $2u - u = u$. As a result u is a multiple of the period c. So u can be defined as the smallest multiple of c greatest than ℓ. We denote $\ell = qc + r$ the Euclidean division of ℓ by c. If $r = 0$ then $u = qc = \ell$. If $r > 0$ then $u = (q+1)c$. And $u = (q+1)c \leqslant (q+1)c + r = qc + r + c = \ell + c$. So the position u is before $c + \ell$ where the first collision takes place. And we hope that $c + \ell$ is a $\mathcal{O}(\sqrt{n})$ (This would be the case if f was really a random function). As a result, with the Floyd method we should find the collision with $\mathcal{O}(\sqrt{n})$ iterations, and a few storage.

A variant of this method (the λ method) gives a parallel algorithm.

To sum up we have first a subfunction:
Function iteration:
input: (X, i, j): s.t. $X = g^i h^j$
output: $F(X), i', j'$: s.t. $F(X) = g^{i'} h^{j'}$

Then the main algorithm:

input: n, g and h
output: x s.t. $h = g^x$
Set i_x, j_x uniformly at random from $\mathbf{Z}/n\mathbf{Z}$
$X = g^{i_x} h^{j_x}$
$(X, i_x, j_x) = \text{iteration}(X, i_x, j_x)$
$(Y, i_y, j_y) = \text{iteration}(X, i_x, j_x)$
While $X \neq Y$:
    $(X, i_x, j_x) = \text{iteration}(X, i_x, j_x)$
    $(Y, i_y, j_y) = \text{iteration}(Y, i_y, j_y)$
    $(Y, i_y, j_y) = \text{iteration}(Y, i_y, j_y)$
If $j_y - j_x$ non invertible modulo n then the algorithm fails
Else return $(i_x - i_y)(j_y - j_x)^{-1} \pmod n$

Pohlig–Hellman

The Pohlig–Hellman method reduces the computation of discrete logarithms in a group of order n to computing logarithm in subgroups of order p where p is prime and p divides n.

We denote $n = p_1^{e_1} \dots p_r^{e_r}$ and $h = g^x$ Then we compute the discrete logarithm modulo the $p_i^{e_i}$ and the value modulo n is computed by the Chinese Remainder Theorem.

So we have to compute x modulo p^e. We denote $x \mod p^e = a_0 + a_1 p + \dots + a_{e-1} p^{e-1}$, where $0 \leqslant a_i \leqslant p-1$. Then from $h = g^x$, we have $h^{n/p} = (g^{n/p})^{a_0}$ and $g^{n/p}$ as order p (g is a generator). So we can compute $a_0 \mod p = a_0$ by an algorithm that computes discrete logarithm modulo p. Then $h^{n/p^2} = (g^{n/p^2})^{a_0 + a_1 p}$, so $(h/g^{a_0})^{n/p^2} = (g^{n/p^2})^{a_0 + a_1 p - a_0} = (g^{n/p^2})^{a_1 p} = (g^{n/p})^{a_1}$. So we can find a_1 by another computation of a discrete logarithm modulo p. By iterating this method, one finds $x \mod p^e$ with e computations of discrete logarithms $\mod p$.

This gives the following algorithm:
Input : n and its factorization: $(p_1, \dots, p_r)$, $(e_1, \dots, e_r)$, g and h

Output : x s.t. $h = g^x$.
For $i = 1$ to r :
Denote $\tilde{g} = g^{n/p_i}$; $\tilde{h} = \tilde{h}^{n/p_i}$; $a_0 = log_{\tilde{g}}(\tilde{h})$; $f = 1$; $x_i = a_0$
For $j = 1$ to $e - 1$:
$f = fg^{a_{j-1}p_i^{j-1}}$; $\tilde{h} = (hf^{-1})^{n/p_i^{j+1}}$
Set $a_j = log_{\tilde{g}}(\tilde{h})$
$x_i = x_i + a_j p_i^j$
Return $x \pmod n$ s.t. $x \equiv x_i \pmod{p_i^{e_i}}$ for $i = 1, \dots, r$.

In practice, for cryptographic applications, one always (almost) takes n prime, otherwise if its factorization is tractable, one can reduce to smaller computation of discrete logarithm modulo $p|n$.

Index Calculus Algorithms

We suppose n to be prime. This is not a generic algorithm: we suppose that there exists $S = \{p_1, p_2, \dots, p_t\} \subset G$ a factor basis s.t. a large proportion of elements of G can be written efficiently as a product of the p_i's.

Pre-computation : We pre-compute the $log_g(p_i)$: we take randoms k in $\mathbf{Z}/n\mathbf{Z}$ and for a large proportion we can write $g^k = \prod p_i^{e_i}$ (easily parallelizable)
By applying the log : $k = \sum e_i log_g(p_i)$. With at least t independant linear equations, we can resolve a linear system that gives the $log_g(p_i)$.

Then, in an active phase, if $h = g^x$, we compute hg^k with random k. If we can factor in S, by applying the log, we have $x + k = \sum e_i log_g(p_i)$.

Trade-off on t : small, we need a small system of t equations. Big, there is a better probability that the random elements can be facored in S. So we make less random choices.

Sub exponential complexity: $L_n(\alpha, c) = \mathcal{O}(exp(c(log\, n)^\alpha (log\, log\, n)^{1-\alpha}))$.
$L(0, c) = (log\, n)^c$: polynomial complexity (in the size of n) and $L(1, c) = n^c$ exponential complexity.

In $\mathbf{Z}/p\mathbf{Z}$ with S = {primes < B} one obtains an algorithm in $L_p(1/2, \sqrt{2})$ (Kraitchik, 1922, rediscovered at the end of the 70's). The number field sieve (NFS, 1993) and the function field sieve (FFS, 1994) are improvement of this idea, they search respectively for smooth elements in function fields and number fields. For computing discrete logarithms in finite fields F_q, these algorithms have an expected time complexity of $L_q(1/3, c)$ for some $c > 0$, FFS being dedicated to small characteristic and NFS to large characteristic.
Starting from 2013, they have been lots of improvements on algorithms to compute discrete logarithm in finite fields F_{p^n}, depending on the size and the form of p and n.
When $n = 1$, *i.e.*, in $\mathbf{Z}/p\mathbf{Z}$, the best algorithm remains the number field sieve. The record is for a 768 bits p, June 2016 (with huge computing power: more than a year, approximately 6600 core years).
For small characteristic, there exists algorithms in $L_q(1/4, c)$ with $c > 0$ and quasi polynomial for very small p, especially when the extension degree n is composite. As a result the finite fields $\mathbf{F}_{2^n}$ must not be used for cryptography, especially pairing-based cryptography with elliptic curve over $\mathbf{F}_{2^n}$. Records: In $F_{2^{9234}}$ january 2014 (400,000 core hours)
In the others cases, there exists many variants of the number field sieve, that improve the asymptotic complexity (still $L_q(1/3, c)$ but with a better c): for example, when p as a special form, or/and when $q = p^n$ and n composite with relatively prime factors (Kim-Barbulescu, 2016). This has an important impact on pairing-based cryptography, for which the discrete logarithm problem has to be hard in F_p^n and a popular choice is $n = 6$ or $n = 12$.
For records, cf. `http://en.wikipedia.org/wiki/Discrete_logarithm_records`

Conclusion

In $(\mathbf{Z}/p\mathbf{Z})^\times$ for instance, to have 128 bits security (*i.e*, the best attack have a complexity of 2^{128} operations), we take p of at least 3072 bits (for NFS) and we work in a subgroup of order q, a prime dividing $p-1$ with q of order 256 bits to avoid attacks with BS/BG and Pollard.

Security level	bitsize of p
80	1024
112	2048
128	3072
192	7680
256	15360

For most of elliptic curves, the index calculus methods can not be applied. So only the exponential generic algorithms can be applied (we will see the situation of pairing friendly curve later).

As a result elliptic curves can be used to define groups where the discrete logarithm problem is intractable with smaller parameters : $\mathbf{Z}/p\mathbf{Z}$, $E(\mathbf{Z}/p\mathbf{Z})$ may define a cyclic group of prime order, with order of roughly the same size of p. With p of $2k$ bits we obtain a k bit security, e.g. 256 bits p for 128 bit security.

Record for elliptic curve With F_p of 112 bits (so the order of 2^{56} computations) in 2009 with a parallelized Pollard ρ and 200 PlayStation 3 in 6 months.

Records over binary fields of roughly the same sizes.

Chapitre II

First Cryptographic Applications

Let (G, x) be a cyclic group, $G = \langle g \rangle$ of prime order n.

1. Diffie-Hellman (76)

Key exchange. With this protocol, Alice and Bob jointly establish a secret quantity over a public channel. This quantity can be used later to derive a secret key used for symmetric encryption.

Alice takes a random $a \in \mathbf{Z}/n\mathbf{Z}$, computes $X = g^a$ and sends X to Bob. Bob takes a random $b \in \mathbf{Z}/n\mathbf{Z}$, computes $Y = g^b$ and sends Y to Alice. Alice and Bob can compute $Z = g^{ab} = Y^a = X^b$.

An eavesdropper can retrieve X and Y. Computing $Z = g^{ab}$ from $X = g^a$ and $Y = g^b$, is called the Computational Diffie Hellman Problem (CDH), and the triplet $(X = g^a, Y = g^b, Z = g^{ab})$ is called a Diffie-Hellman triplet.

Distinguishing Z from a random element of G, knowing X and Y, is called the Decisional Diffie–Hellman (DDH).

In general, the only known way to resolve these problem is by computing one of the discrete logarithm (e.g., to compute a from X, and retrieve $Z = Y^a$).

2. Elgamal Encryption (84)

Asymmetric encryption scheme. Can be viewed as a Diffie-Hellman key exchange, followed by a one time pad in G: the symmetric encryption scheme, that encrypts $m \in G$ as mz where z is a secret key used only once.

The element Y is viewed as Bob's public key: $pk = Y = g^b$. To encrypt $m \in G$ for Bob, Alice sends $X = g^a$ and $mY^a = mZ$. Bob can decrypts thanks to b, which is his secret key: from b and X he can compute Z and recover m.

More precisely, with a little change of notations, the Elgamal encryption scheme can be described as follows:

- $pk = h = g^x$, where x is random in $\mathbf{Z}/n\mathbf{Z}$ and $sk = x$.
- Encryption of $m \in G$ with the public key $h = pk$: takes a random $r \in \mathbf{Z}/n\mathbf{Z}$ and $c = (c_1, c_2) = (g^r, mh^r)$.
- Decryption of $c = (c_1, c_2)$ with the private key $x = sk$: Compute $c_2(c_1^x)^{-1}$

The scheme is correct as if $(c_1, c_2) = (g^r, mh^r)$ then $c_2(c_1^x)^{-1} = mh^r \times (g^{rx})^{-1} = mg^{rx} \times (g^{rx})^{-1} = m$. Note that it is a probabilistic encryption scheme.

It is easy to see that recovering the secret key from the public key is an hard problem if the discrete logarithm problem is hard in G. One can also prove that recovering m from c and pk (breaking the One-Wayness

of encryption) is hard if the CDH problem is hard and moreover that finding any partial information of m from c and pk (breaking semantic security) is hard if the DDH is hard.

3. Digital Signature

3.1. Definition

Digital Signatures are roughly equivalent to handwritten signatures on paper documents.

Notion of non repudiation: someone that has signed some information can not deny having signed it (only the person that knows the private key can have produce the signature, it can not be forged). As a result, it is a way to authenticate the authors of a message. Moreover, the signed message can not be modified (or otherwise the signature is invalid): notion of integrity. The signature can be verified by anyone with a public key.

More precisely, a digital signature scheme works as follows:

- Alice has pk a public key (for verification) and sk a private key (for signing).
- With a message m and her private key sk, Alice executes a signing algorithm to get σ the signature.
- With a message m, the signature σ of m by Alice, and pk the public key of Alice, Bob executes a verification algorithm to verify if the signature is valid.

In general, the signing algorithm uses first a cryptographic hash function H applied on the message. This is a function H from $\{0,1\}^* \to$ A where A is a finite set, for instance A $= \{0,1\}^n$. Moreover H has several property (one-way, collision resistance,...). The hash function makes digital signature more practical (can be applied to any digital files), and also provides security.

3.2. Elgamal Signature (84)

We give here a generalization of the signature scheme of Elgamal in a cyclic group G (original scheme in $(\mathbf{Z}/p\mathbf{Z})^\times$).

A message m, is mapped to M = H(m) $\in \mathbf{Z}/n\mathbf{Z}$ where H is an hash function H : $\{0,1\}^* \to \mathbf{Z}/n\mathbf{Z}$.

A first idea (not working) is to use the same setup as in the Elgamal encryption scheme: The public key is $h = g^x$ and the private key is $x \in \mathbf{Z}/n\mathbf{Z}$. We try to build a signature of m from a Diffie-Hellman triplet that involves M. We thus consider an Elgamal encryption of g^{M} : $(g^r, h^r g^{\mathrm{M}}) = (g^r, g^{xr+\mathrm{M}})$ with r random in $\mathbf{Z}/n\mathbf{Z}$. Let $s = xr + \mathrm{M} \in \mathbf{Z}/n\mathbf{Z}$ and $f = g^r$, we then have $(g^r, g^{xr+\mathrm{M}}) = (f, g^s)$. The signature of m could be $\sigma = (f, s)$, and the verification could consists in verifying if $(h, f, g^s/g^{\mathrm{M}}) = (g^x, g^r, g^{xr})$ is a Diffie-Hellman triplet.

Issues: The only (generic) known to test if this is a Diffie-Hellman Triplet, is by knowing one of the discrete logarithms (r or x but this is the secret key). Knowing r, the scheme is trivially insecure: from r, s, m one can compute x as $x = r^{-1}(s - \mathrm{M}) \in \mathbf{Z}/n\mathbf{Z}$ if r is invertible ($r \neq 0$ as n is prime).

Solution: We will see later that with elliptic curve pairings it is possible to distinguish if we have a Diffie-Hellman Triplet or not, without knowing individual discrete logarithm. For now, without pairings, the solution used by Elgamal is to use H(f) instead of using r to compute s, and instead of having $g^s = g^{xr+\mathrm{M}}$, he defines s such that $f^s = g^{x\mathrm{H}(f)+\mathrm{M}}$. As a result, the equation for the exponent is $rs = x\mathrm{H}(f) + \mathrm{M}$.

To summarize, the protocol is as follows:

- G of order n and H a hash function $\{0,1\}^* \to \mathbf{Z}/n\mathbf{Z}$.
- $pk = h = g^x$ with a random x, $sk = x$
- Signing a message m with the key x: r a random invertible element modulo n, $f = g^r$, $s \equiv r^{-1}(x\mathrm{H}(f) + \mathrm{H}(m)) \pmod n$. $\sigma = (f, s)$.

- Verifying a signature $\sigma = (f, s)$ of m with the public key h: $v_1 = f^s$, $v_2 = h^{\mathrm{H}(f)}g^{\mathrm{H}(m)}$, ok if $v_1 = v_2$.

The scheme is correct has $v_1 = f^s = g^{rs} = g^{x\mathrm{H}(f)+\mathrm{H}(m)}$ and $v_2 = h^{\mathrm{H}(f)}g^{\mathrm{H}(m)} = g^{x\mathrm{H}(f)+\mathrm{H}(m)}$.

DSA: Digital Signature Algorithm (NIST 91) is a variant of the Elgamal signature in a subgroup of prime order q of $(\mathbf{Z}/p\mathbf{Z})^\times$.

4. Elliptic Curve Cryptography

Elliptic curve are mostly used in cryptography for

- key exchange: ECDH which is an adaptation of the Diffie-Hellman key exchange in the group of points of an Elliptic curve
- Signature: ECDSA, an adaptation of the Elgamal signature, see details below.

There are lots of others applications that use pairings of elliptic curves. ECDH and ECDSA are standardized. They can be used to secure communications over a computer network with Transport Layer Security (TLS), since version 1.0 (1999). There are more and more used (see Google certificate for instance). Indeed, the common alternatives to these two protocols are respectively the Diffie-Hellman key exchange in finite fields and the RSA signature scheme which is based on the hardness of factoring integers. Nowadays, both protocols need key sizes of at least 2048 bits and 3072 bits is recommended.

For elliptic curve, one can work with smaller key sizes. An elliptic curve defined on a finite field of size around 200 to 256 bits is recommended. As we saw in the previous chapter, the best algorithm to compute discrete logarithm for elliptic curve (the ρ method) is exponential, and the best algorithm to compute discrete logarithm for finite fields and factoring integers (the number field sieve) is sub-exponential. As a result, with the increasing power of computers, elliptic curves become more and more interesting, especially for embedded systems such as smart cards, where it is crucial to work with small data.

4.1. ECDSA

This is a variant of the Elgamal signature, with some changes and optimization to handle the representation of the group elements, *i.e.*, the points of an elliptic curve.

- We work in a cyclic subgroup $\langle \mathrm{P} \rangle$ of prime order n of the groups of points of an elliptic curve. Let H be an hash function $\{0,1\}^*$ with values in $\{1, \dots, n-1\}$. In practice, the function SHA2 is used with a truncation of the output to get ℓ bits where ℓ is the number of bits of n
- $pk = \mathrm{Q} = x\mathrm{P}$ with x random $0 < x < n$, $sk = x$
- Signing a message m with the key x: r random, $0 < r < n$, $\mathrm{R} = (x_\mathrm{R}, y_\mathrm{R}) = r\mathrm{P}$, If $x_\mathrm{R} \mod n = 0$, choose an other r. $s \equiv r^{-1}(x(x_\mathrm{R} mod n) + \mathrm{H}(m)) \pmod n$. If $s \equiv 0$ choose an other r. The signature is $\sigma = (\sigma_1, \sigma_2) = (x_\mathrm{R} \mod n, s)$.
- Verifying a signature (σ_1, σ_2). Verify that Q is on the curve, and that Q has order n and that $1 < \sigma_i < n$, for $i = 1, 2$. Then compute $u_1 \equiv \mathrm{H}(m)\sigma_2^{-1} \pmod n$ and $u_2 \equiv \sigma_1\sigma_2^{-1} \pmod n$, and then $(x_1, y_1) = u_1\mathrm{P} + u_2\mathrm{Q}$. ok if $\sigma_1 \equiv x_1 \mod n$.

The scheme is correct as $u_1\mathrm{P} + u_2\mathrm{Q} = (u_1 + u_2 x)\mathrm{P} = (\mathrm{H}(m)s^{-1} + (x_\mathrm{R} mod n)s^{-1}x)\mathrm{P} = s^{-1}(\mathrm{H}(m) + (x_\mathrm{R} mod n)x)\mathrm{P} = r(\mathrm{H}(m) + x(x_\mathrm{R} mod n))^{-1}(\mathrm{H}(m) + (x_\mathrm{R} mod n)x)\mathrm{P} = r\mathrm{P}$.

The differences with Elgamal signature scheme, is that only x_R is given in the signature, instead of $r\mathrm{P}$, and that this point is not hashed, x_R is used instead.

Note that r must be random. If two signatures are issued with the same r: σ, σ' of m and m'. Then, $\sigma_2 - \sigma_2' = r^{-1}(\mathrm{H}(m) - \mathrm{H}(m'))$. So r can be recovered. Then as $\sigma_2 = r^{-1}(x(x_\mathrm{R} mod n) + \mathrm{H}(m))$ one can deduce x.

This textbook attack was surprisingly used real life. An attack against the PlayStation 3 in 2010 allows to recover the key used to sign the files that the PlayStation could execute (r was constant, $r = 4$), Another attack on a digital wallet for Bitcoins on Android was done in 2013. The application was using a weak random number generator (ECDSA is used in the Bitcoin protocol to prove the property of Bitcoins). As a consequence, some bitcoins were stolen.

4.2. How to generate an elliptic curve suitable for cryptography

FIPS 186-4: 2013, 4th revision, US standard describes ECDSA cf. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`

FIPS gives also some standardized curves and a standardized way to generate a random elliptic curve.

Standard Elliptic Curves

Curve over $\mathbf{F}_p$, p odd prime. We will work with $G = \langle P \rangle$ where $P = (x_P, y_P)$ has order n and n is prime. Let h such that $Card\, E = nh$. This co factor h must be small, to improve efficiency: n will be of roughly the same bit size than p.

The NIST gives several standard curve over $\mathbf{F}_p$: e.g., $P-192, P-224, P-256, P-384, P-521$, where the number is the bit size of p. The co factor $h = 1$ for these curves (the group of points are cyclic of prime order). The curve equations are $y^2 = x^3 - 3x + b \mod p$. Note: $a = -3$ for efficiency, only a is used in the classic formulae for the group law.

The chosen p have a sparse binary representation in order to speedup computations modulo p.

There are also curves standardized over F_{2^d}:

- 3 Koblitz curves, $K-163, K-233, K-283, K-409, K-571$ where the number is the degree d. Equations $y^2 + xy = x^3 + ax^2 + 1$ where $a = 0, 1$. For these curves, $h = 2$ if $a = 1$ and $h = 4$ if $a = 0$. For these curves, one can explicitly compute the number of points, and there is an algorithm to compute exponentiations without doubling operations.
- 3 standard curves of equations $y^2 + xy = x^3 + x^2 + b$ with $b \in F_{2^d}$, for which $h = 2$. $B-163, B-233 B-283, B-409, B-571$.

Random curves

Modulo $p > 3$ one wants an equation $y^2 = x^3 + ax + b$ with $\Delta = -16(4a^3 + 27b^2) \neq 0 \pmod p$.

To generate a and b, one choses a random seed and apply (several times) an hash function to come up with an integer c. As the hash function is one-way (pre-image can not be computed), this procedure is meant to ensure the users of the generated curve that this curve has been truly generated randomly, and it does not contains an hidden weakness or a trapdoor that could be used to recover the private keys...

Then a and b are chosen such that $cb^2 \equiv a^3 \pmod p$ (e.g., $a = c$, $b = c$ but one can take a small a to speed up computations). This is to generate isomorphism classes of elliptic curve (two elliptic curves of parameters a, b and a', b' over a finite field with characteristic at least 5 are isomorphic if and only if $u^4 a' = a$ et $u^6 b' = b$. As a result, the value of $c = a^3/b^2 = a'^3/b'^2$ is fixed inside a class.

Note that $\Delta \neq 0 \Leftrightarrow 4a^3 + 27b^2 \neq 0 \Leftrightarrow -4a^3/27b^2 \neq 1 \Leftrightarrow -4c \neq 27 \Leftrightarrow 4c + 27 \neq 0$.

Then one verifies if the order is suitable, *i.e*, if it has a large prime factor (for that, one can compute the order with an early abort strategy if it is divisible by to much small primes). If it is not the case another seed is taken. There are more advanced criteria to generate a safe curve (e.g., see `https://safecurves.cr.yp.to`).

Another way to generate a curve is to use the complex-multiplication (CM) method. The order is selected first, then an elliptic curve and a finite field is found that meet that order. Over $\mathbf{F}_p$, this method is faster than the algorithms to compute the number of points.