Université de Limoges

Projet scientifique

Master 1 Cryptis

La vulnérabilité de l'ECDSA

Réalisé par:
Hajar Boudfor
Ibtissam Bahadou

Encadrant:
Simone Naldi

Mai,2023

Table des matières

1 Introduction

La cryptographie joue un rôle crucial dans la sécurisation des communications et des transactions en ligne, en particulier dans le domaine des crypto-monnaies. Parmi les diverses méthodes de cryptographie existantes, l'algorithme ECDSA (Elliptic Curve Digital Signature Algorithm) est largement utilisé pour garantir l'intégrité et l'authenticité des transactions numériques. Au cours du premier semestre, notre projet de recherche s'est concentré sur l'étude de la cryptographie appliquée aux crypto-monnaies, avec une attention particulière portée à l'algorithme ECDSA.

Dans ce rapport, nous abordons le deuxième volet de notre projet, qui consiste à étudier les vulnérabilités potentielles de l'algorithme ECDSA. L'objectif de cette recherche est d'identifier et d'analyser les faiblesses éventuelles de cet algorithme largement utilisé .Nous avons étudié diverses sources pour mieux comprendre les attaques potentielles visant l'ECDSA, notamment une étude de cas sur une attaque spécifique appelée "nonce et impact sur ECDSA". Ce rapport synthétise les informations clés tirées de ces recherches.

2 Standard de signature numérique

2.1 Motivation

L'algorithme de signature numérique (DSA) a été développé par le National Institute of Standards and Technology (NIST) au milieu des années 1990 dans le cadre de la norme de signature numérique (DSS) afin de fournir des signatures numériques sécurisées.
Il est basé sur le problème mathématique du calcul des logarithmes discrets, qui est considéré comme un problème informatiquement difficile. Plus précisément, DSA utilise des sous-groupes d'ordre premier de $\mathbb{F}_{Z_p}$ pour assurer la sécurité cryptographique, où p est un grand nombre premier.

En plus de fournir une sécurité cryptographique, l'DSA est également conçu pour être relativement efficace en termes de complexité de calcul. Cela en fait un choix populaire pour de nombreuses applications où la vitesse et l'efficacité sont des considérations importantes.
Toutefois, il est important de noter que DSA présente certaines limites, notamment en termes de taille de clé et de potentiel pour certains types d'attaques.

Malgré ces limites, DSA reste un outil important pour garantir l'authenticité et l'intégrité des documents numériques. Il est largement utilisé dans une variété d'applications, y compris les transactions électroniques, les protocoles de communication sécurisés et les certificats numériques. Alors que le paysage numérique continue d'évoluer, l'importance des signatures numériques sécurisées ne fera que croître, et l'DSA restera un élément essentiel de l'infrastructure de sécurité numérique.

2.2 Génération de paramètres de domaine DSA

Génération de paramètres de domaine DSA.
Les paramètres de domaine sont générés pour chaque entité d'un domaine de sécurité particulier.

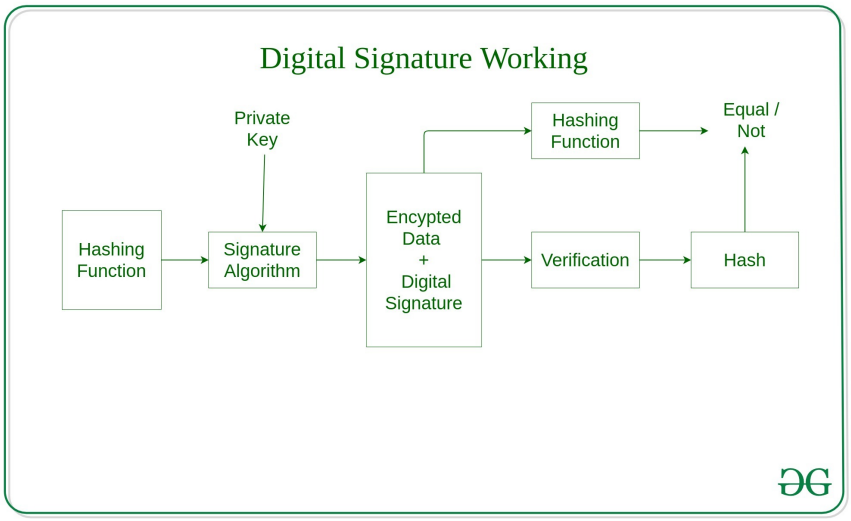


Figure 1: *shéma géneral d'algorithmes de signatures*

1. Sélectionnez un nombre premier de 160 bits q et un nombre premier de 1024 bits p avec la propriété que $q|p-1$.
2. (Choisir un générateur g de l'unique groupe cyclique d'ordre q dans $\mathbb{Z}_p^*$.)
3. Choisissez un élément $h \in \mathbb{Z}_p^*$ et calculez $g = h^{(p-1)/q} \bmod p$. (Répétez jusqu'à ce que $g \neq 1$.)
4. Les paramètres du domaine sont p, q et g.

2.3 Génération de paires de clés DSA

Génération de paires de clés DSA.
Chaque entité A du domaine avec des paramètres de domaine (p, q, g) effectue les opérations suivantes :

1. Sélectionner un entier aléatoire ou pseudo-aléatoire x tel que $1 \leq x \leq q-1$.
2. Calculer $y = g^x \bmod p$.
3. La clé publique de A est y ; la clé privée de A est x.

2.4 Génération de signature DSA

Génération de signature DSA.
Pour signer un message m, A effectue les opérations suivantes :

1. Choisir un nombre entier aléatoire ou pseudo-aléatoire k, $1 \leq k \leq q-1$.

2. Calculer $X = g^k \text{mod } p$ et $r = X \text{mod } q$. Si $r = 0$, retourner à l'étape 1.

3. Calculer $k^{-1} \text{mod } q$.

4. Calculer $e = SHA - 1(m)$.

5. Calculer $s = k^{-1}(e + xr) \text{mod } q$. Si $s = 0$, retourner à l'étape 1.

6. La signature de A pour le message m est (r, s).

2.5 Vérification de la signature DSA.

Vérification de la signature DSA.
Pour vérifier la signature de A (r, s) sur m, B obtient des copies authentiques des paramètres du domaine de A (p, q, g) et de la clé publique y et procède comme suit :

1. Vérifier que r et s sont des entiers dans l'intervalle $[1, q - 1]$.

2. Calculer $e = SHA - 1(m)$.

3. Calculer $w = s^{-1} \text{ mod } q$.

4. Calculer $u_1 = ew \text{ mod } q$ et $u_2 = rw \text{ mod } q$.

5. Calculer $X = g^{u_1} y^{u_2} \text{mod } p$ et $v = X \text{mod } q$.

6. Accepter la signature si et seulement si $v = r$.

3 Fonctionnement ECDSA :

3.1 Algorithme ECDSA :

L' algorithme ECDSA utilise un ensemble fini de points (x, y) où x et y sont entiers et vérifient une équation du type :

$$y^2 = (x^3 + a * x + b)[modn]$$

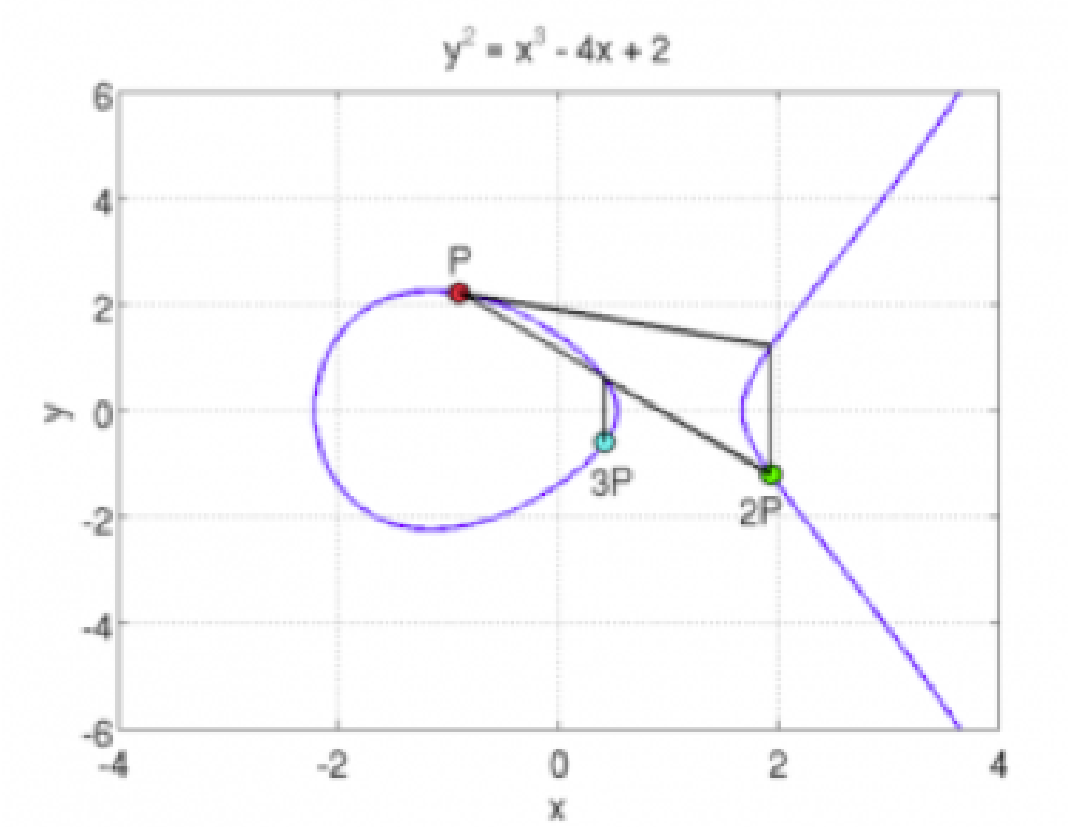
ECDSA utilise par ailleurs une loi de groupe qu'on pourrait appeler l'« addition » de deux points P et Q sur la courbe, ainsi définie: $P + Q = R$ quand on trace une droite entre P et Q qui coupe la courbe au point S et quand R est le symétrique de S sur la courbe, par rapport à l'axe des x.

Quand P et Q se rapprochent, la droite PQ tend vers la tangente à la courbe, donc on peut additionner un point avec lui-même en prenant cette tangente au point P pour construire les points S et R.

En effet : L'addition de deux points sur une courbe elliptique se fait par la construction d'une droite passant par ces deux points et par l'intersection de cette droite avec la courbe. Cette intersection peut donner lieu à un troisième point qui est le résultat de l'addition des deux points initiaux.

Dans le cas où l'on souhaite additionner un point P à lui-même, il n'est pas possible de construire une droite passant par deux points distincts, car P est un point unique. Pour résoudre ce problème, on utilise la propriété mathématique de la tangente à la courbe elliptique en un point P.

La tangente à la courbe elliptique en un point P est une droite qui ne passe que par ce point, et qui est définie comme la droite qui touche la courbe en ce point. On peut alors utiliser cette tangente pour définir l'addition du point P avec lui-même.



On a alors$R = 2P, 3P = R + P$ et ainsi de suite de sorte qu'on peut calculer un point kxP où k est un entier quelconque, tant que R ne se trouve pas à l'infini

3.2 Comment fonctionne la vérification de signature ECDSA pour assurer la sécurité de Bitcoin ?

Pour signer un message, l'empreinte M du message est calculée avec une fonction de hachage (SHA1), qui produit un condensé de 20 octets (160 bits). Ensuite, pour vérifier la signature, il est nécessaire de retrouver l'entier M à partir du point$C = MxG$ (la signature) et du point de base G.MxG signifie que le point G est « additionné » avec lui-même M fois, l'addition de deux points définissant une loi de groupe sur l'ensemble des points d'un champ fini.

Le champ fini comporte un grand nombre de points et, si le point de base G est bien choisi, l'attaque « brute force » consistant à calculer tous les points kxG pour toutes les valeurs possibles de k (entier) et à vérifier que l'un de ces points coïncide avec C est impraticable : c'est là que réside la sécurité de ECDSA et donc de bitcoin.

En effet : Le point de base G est un point fixé à l'avance sur la courbe elliptique et utilisé pour générer les clés publiques et privées nécessaires à la signature. Si le point de base G est bien choisi, alors il est possible de générer un grand nombre de points différents en multipliant ce point par différents entiers k. Cependant, le nombre de points possibles sur la courbe elliptique est très grand, ce qui rend impraticable une attaque brute force consistant à essayer toutes les valeurs possibles de k pour trouver le point correspondant à la signature C.

En d'autres termes, même si un attaquant connaît le point C et le point de base G, il est très difficile de retrouver l'entier k utilisé pour générer la signature à partir de l'équation$C = kxG$. Cette difficulté est liée à la structure mathématique de la courbe elliptique et à la complexité du problème du logarithme discret dans les groupes de points de courbes elliptiques. C'est cette difficulté qui rend la signature ECDSA utilisée dans Bitcoin et d'autres cryptomonnaies sécurisée et résistante aux attaques.

3.3 les paramètres d'une courbe elliptique utilisée pour la signature ECDSA

Les paramètres de la courbe doivent être choisis avec soin pour éviter d'utiliser une courbe faible.La courbe utilisée par bitcoin est une courbe dite de Koblitz du nom du cryptographe, Neal Koblitz, qui, en 1985, a démontré l'utilité des courbes elliptiques en cryptographie.

Zp: champ fini premier où p est un grand nombre premier (p = caractéristique du champ, champ étant un cas particulier d'un anneau)

On considère les points (x,y) où x et y sont dans Zpet vérifient :

$$y^2 = (x^3 + a * x + b)[modn]$$

Le nombre de ces points constitue ce qu'on appelle l'ordre de la courbe.

n: ordre du point G, plus petit entier tel que n x G = O (O : élément «identité» du groupe fini). G doit être choisi de sorte que n est un grand nombre entier.

h: cofacteur, l'ordre de la courbe elliptique divisé par n.

k: à chaque signature, le signataire va choisir au hasard un entier k, très grand mais strictement inférieur à n, pour fabriquer sa signature. Noter que cet entier k ne sera pas nécessaire pour vérifier la signature et que k devra être changé à chaque signature.

3.4 Génération des clés ECDSA:

La première étape de la fabrication d'une signature ECDSA consiste donc à choisir un entier X, strictement plus petit que n, comme clé privée puis à calculer la " clé publique " Y telle que : $Y = X * G$

La clé publique Y est donc en fait un point sur la courbe. Y ainsi que les paramètres p, n, a, b et G doivent être publiés.

3.5 Calcul et vérification d'une signature ECDSA :

Après avoir calculé le condensé M du message avec la fonction de hachage SHA-1, la signature se compose de deux parties, sig1 et sig2.

Tout d'abord, on calcule sig1 en multipliant un nombre aléatoire k par le point G (k×G) et on ne retenant que sa coordonnée x (modulo n) :

$$sig1 = (k * G)x[modn]$$

Si le calcul du modulo donne zéro, il faut recommencer avec une autre valeur de k. Ensuite, on calcule $sig2$ en utilisant la formule suivante :

$$sig2 = k^{-1} * (M + X * sig1)[modn]$$

où k^{-1} est l' inverse de k modulo n.

Le destinataire qui reçoit le message et la signature (sig1, sig2) peut vérifier l'authenticité du message en calculant d'abord son condensé M avec SHA-1, puis les nombres :

$w = sig2^{-1}[modn]$
$u1 = M * w[modn]$
$u2 = sig1 * w[modn]$

Enfin, en calculant le point : $(x, y) = u1ÖG + u2ÖY$, la signature est vérifiée si $sig1 = x[modn]$.

Exemple 1 :

Dans cet exemple, nous allons illustrer le processus de création et de vérification d'une signature ECDSA pour une transaction Bitcoin en utilisant la courbe elliptique secp256k1.

3.5.1 Génération des clés

Tout d'abord, un utilisateur doit générer une paire de clés, composée d'une clé privée (d) et d'une clé publique (Q). La courbe elliptique secp256k1 et son point de base G sont utilisés pour la génération des clés. La clé privée est un nombre aléatoire choisi entre 1 et l'ordre de la courbe moins 1. La clé publique est obtenue en multipliant la clé privée par le point de base$G(Q = dÖG)$.

3.5.2 Signature du message :

Supposons que nous ayons une transaction Bitcoin simple à signer. Le message est d'abord haché en utilisant l'algorithme SHA-256, produisant un condensé M du message. Ensuite, un nombre aléatoire k est généré, qui est utilisé pour calculer le point P sur la courbe elliptique$(P = kÖG)$. La première partie de la signature, sig1, est obtenue en prenant la coordonnée x de P et en la réduisant modulo n (l'ordre de la courbe). Si sig1 est égal à zéro, un nouveau nombre aléatoire k doit être généré et le processus répété. La deuxième partie de la signature, sig2, est calculée en utilisant la formule : $sig2 = k^{-1} * (M + d * sig1)$ [mod n], où k^{-1} est l'inverse de k modulo n.

La signature de la transaction Bitcoin est représentée par le couple (sig1, sig2).

Pour vérifier la signature, le destinataire doit d'abord calculer le condensé M du message (la transaction) en utilisant l'algorithme SHA-256. Ensuite, les valeurs intermédiaires w, u1 et u2 sont calculées en utilisant les formules suivantes :

$w = sig2^{-1}[modn]$
$u1 = M * w[modn]$
$u2 = sig1 * w[modn]$

Le point (x, y) sur la courbe elliptique est ensuite calculé en utilisant l'équation: $(x, y) = u1ÖG + u2ÖQ$.
La signature est considérée comme valide si sig1 est égal à la coordonnée x du point (x, y) calculé précédemment, modulo n ($sig1 = x[modn]$).

Exemple 2 :

Dans cet exemple, nous utilisons le corps fini F_{23} et une courbe elliptique définie par l'équation $y^2 = x^3 + x + 1$. Nous choisissons le point $G = (13, 7)$ sur la courbe, qui a un ordre $n = 7$ (car $7G = 0$). Les paramètres de domaine publics sont le corps F_{23}, la courbe E, le point G, l'ordre $n = 7$ et le cofacteur $h = 4$.

1.Génération de clés: L'entité A procède comme suit pour générer ses clés :

1. A sélectionne un entier aléatoire $d = 3$ dans l'intervalle $[1, n - 1]$, c'est-à-dire $[1, 6]$.
2. A calcule ensuite le point $Q = dG$, soit $3(13, 7)$, qui donne $(17, 3)$.
3. A publie le point Q, qui constitue sa clé publique.
4. La clé privée de A est l'entier $d = 3$.

2.Génération de signature avec ECDSA : Pour signer le message $M = 11100011010111100$, l'entité A procède comme suit :

1. Supposons que la représentation décimale de la valeur de hachage $H(M)$ soit $e = 6$.
2. Sélectionne un entier aléatoire $k = 4$ dans l'intervalle $[1, n - 1]$, soit $[1, 6]$.
3. Calcule $(x_1, y_1) = kG = 4(13, 7) = (17, 20)$.
4. Représente x_1 comme l'entier $v_1 = 17$.
5. Définit $r = x_1 \mod n = 17 \mod 7 = 3$.
6. Calcule $s = k^{-1}(e + dr) \mod n = 4^{-1}(6 + 3 \times 3) \mod 7 = 2(15) \mod 7 = 2$.
7. La signature du message M est $(r, s) = (3, 2)$.

3.Vérification de la signature avec ECDSA : L'entité B vérifie la signature $(r', s') = (3, 2)$ sur M comme suit :

1. B récupère la clé publique de A, $Q = (17, 3)$.
2. Calcule $e' = 7$, la représentation décimale de $H(M)$.
3. Calcule $c = (s')^{-1} \mod n = 2^{-1} \mod 7 = 4$.
4. Calcule $u_1 = e'c \mod n = 6 \times 4 \mod 7 = 3$ et $u_2 = r'c \mod n = 3 \times 4 \mod 7 = 5$.
5. Calcule le point $(x', y') = u_1G + u_2Q = 3G + 5Q = 3(13, 7) + 5(17, 3) = (17, 20)$.
6. Représente x' comme l'entier $t = 17$.
7. Calcule $v = x' \mod n = 17 \mod 7 = 3$.
8. Accepte la signature puisque $v = r' = 3$

4 Attaques génériques sur l'ECDSA

L'objectif de sécurité d'ECDSA est d'être inviolable contre une attaque par messages choisis. Le but d'un adversaire qui lance une telle attaque contre une entité légitime A est d'obtenir une signature valide sur un seul message m, après avoir obtenu la signature d'A sur une collection de messages (à l'exclusion de m) choisis par l'adversaire.

Soit A un utilisateur qui utilise un système de cryptographie à clé publique avec une paire de clés publique/privée (Q, d). Les paramètres de domaine de A sont définis comme suit:

- q: un nombre premier qui définit le corps fini sur lequel repose le système.
- FR: une description des fonctions de hachage et de chiffrement utilisées par le système.
- a, b: deux entiers qui définissent une courbe elliptique E utilisée par le système.
- P: un point de la courbe elliptique E qui génère un sous-groupe cyclique de la courbe elliptique E.
- n: un entier qui représente l'ordre de P.
- h: un entier qui représente le cofacteur de P.

Un adversaire peut réussir à falsifier la signature d'A sur n'importe quel message en calculant la clé privée d de A à partir de ses paramètres de domaine (q, FR, a, b, P, n, h) et de sa clé publique Q. Cela peut être représenté mathématiquement comme suit:

$$d = f(q, FR, a, b, P, n, h, Q)$$

où f est une fonction qui calcule la clé privée d de A à partir des paramètres de domaine et de la clé publique Q de A. Une fois que l'adversaire a calculé la clé privée d, il peut utiliser cette clé pour créer des signatures valides pour n'importe quel message de son choix.

4.1 le problème du logarithme discret sur une courbe elliptique

L'algorithme de signature numérique ECDSA est vulnérable aux attaques basées sur le problème du logarithme discret sur une courbe elliptique. Plus précisément, si l'on considère une courbe elliptique E définie sur un corps fini $\mathbb{F}_q$ et un point de base P appartenant à $E(\mathbb{F}_q)$, la sécurité de l'ECDSA repose sur la difficulté de calculer un logarithme discret dans le sous-groupe engendré par P.

Soit E une courbe elliptique définie sur un corps fini $\mathbb{F}_q$. Soit $P \in E(\mathbb{F}_q)$ un point d'ordre n, où n est un nombre premier et $n > 2^{160}$.

Définition .1. Le problème du logarithme discret sur une courbe elliptique (ECDLP) consiste à déterminer l, étant donné une courbe elliptique E définie sur un corps fini F_q, un point $P \in E(F_q)$ d'ordre n, et un point $Q = lP$ où $0 \leq l \leq n-1$.

4.2 L'attaque de Pohlig-Hellman

L'algorithme de Pohlig-Hellman a été présenté par Stephan C. Pohlig et Martin E. Hellman en 1978. Dans l'article original, il est présenté comme un algorithme amélioré utilisé pour calculer les logarithmes discrets sur le champ cyclique $G = GF(p)$, et comment leurs résultats impactent la cryptographie à courbes elliptiques. Étant donné le problème du logarithme discret d'ECDLP:

$$Q = lP$$

l'algorithme de Pohlig-Hellman est un algorithme **récursif** qui réduit le problème en calculant les logarithmes discrets dans les sous-groupes d'ordre premier de $< P >$. Chacun de ces sous-problèmes plus petits peut ensuite être résolu en utilisant des méthodes telles que l'algorithme de Pollard-Rho.

L'algorithme de Pohlig-Hellman fonctionne comme suit:

1. Faire la décomposition en facteurs premiers n(ie. sous forme $n = p_1^{e_1} p_2^{e_2} ... p_r^{e_r}$
2. Calculer $l_i = l \bmod p_i^{e_i}$ pour tout $1 \leq i \leq r$
   Le théorème des restes chinois est ensuite utilisé pour obtenir une solution unique au système suivant de congruences :

$$\begin{aligned} l &\equiv l_1 \pmod{p_1^{e_1}} \\ l &\equiv l_2 \pmod{p_2^{e_2}} \\ &\vdots \\ l &\equiv l_r \pmod{p_r^{e_r}} \end{aligned}$$

   Ici,$p_1, p_2, ..., p_r$ est un ensemble mutuellement premier d'entiers positifs, c'est-à-dire que $\text{pgcd}(p_i, p_j) = 1$ pour tout $i = j$.
   $l_1, l_2, ..., l_r$ sont tous des entiers positifs tels que $0 \leq l_i < p_i$. Le nombre entier positif unique l peut être calculé efficacement en utilisant l'algorithme d'Euclide étendu pour résoudre les

congruences linéaires ci-dessus.
Chaque l_i peut s'exprimer dans la base p comme suit:

$$l_i = z_0 + z_1 p + z_2 p^2 + ... + z_{e-1} p^{e-1} où\ z_i \in [0, p-1] \tag{1}$$

Soit :

$$P_0 = \frac{n}{p_i} \tag{2}$$

et

$$Q_0 = \frac{n}{p_i} Q \tag{3}$$

En réécrivant les équations, et en utilisant le fait que l'ordre de P_0 est p, on obtient

$$Q_0 = lP_0 = z_0 P_0 \tag{4}$$

Ensuite, z_0 est trouvé en calculant la solution ECDLP dans $\langle P_0 \rangle$. Chaque $z_0, \ldots, z_{e-1}$ est alors calculé en résolvant

$$Q_i = z_i P_0 \tag{5}$$

dans $\langle P_0 \rangle$.

Exemple
Soit la courbe elliptique $E : y^2 = x^3 + 1001x + 75 \pmod{7919}$ avec un ordre de la courbe de $m = 7889$. Soit $P = (4023, 6036)$ et $Q = (4135, 3169)$ deux points de la courbe tels que $xP = Q$. Trouver x.

Solution :

- **Résolution dans G_1:** On décompose $m = 7889$ en facteurs premiers, soit $m = 73 \times 23$.En résolvant dans le sous-groupe 7^3. On multiplie les deux côtés du DLP par $\frac{m}{7}$, on remplace x par x_1

  $$\frac{m}{7} * Q = x_1 \frac{m}{7} * P$$

  On développe x_1 en base 3 :

  $$x_1 = a_0 + 7a_1 + 7^2 a_2$$

  On remplace ceci dans l'équation et on supprime tous les termes après a_0 en utilisant le théorème de Lagrange :

  $$1127Q = (1127a_0)\, P$$

  où $a_0 \in \{0, 1, 2, 3, 4, 5, 6\}$. Il n'y a qu'une seule variable ici, a_0. On peut la résoudre pour obtenir :

  $$\begin{aligned} a_0 &= 1 \\ Q &= \left(1 + 7a_1 + 7^2 a_2\right) \cdot P \\ Q - P &= \left(7a_1 + 7^2 a_2\right) \cdot P \end{aligned}$$

On multiplie les deux côtés par $\frac{m}{7^2}$:

$$\frac{m}{7^2}(Q-P) = \left(7a_1 + 7^2 a_2\right) \cdot \frac{m}{7^2} P$$

Encore une fois, en utilisant le théorème de Lagrange, on supprime le terme 7^2 :

$$161(Q-P) = 1127 a_1 P$$

On résout cela pour obtenir $a_1 = 3$.

$$\begin{aligned} Q &= \left(1 + 7 \times 3 + 7^2 a_2\right) \cdot P \\ Q &= \left(22 + 7^2 a_2\right) \cdot P \\ Q - 22P &= 7^2 a_2 P \end{aligned}$$

On multiplie les deux côtés par $\frac{m}{7^3}$:

$$\begin{aligned} \frac{m}{7^3}(Q-22P) &= 7^2 a_2 \cdot \frac{m}{7^3} P \\ 23(Q-22P) &= 1127 a_2 P \end{aligned}$$

En résolvant, on obtient $a_2 = 4$. Donc, $x_1 = 1 + 7 \times 3 + 49 \times 4 = 218$.
$x \equiv 218 \pmod{7^3}$

- **Résolution dans G_2:** La résolution de ce problème se fait de manière similaire en utilisant le sous-groupe G_2 $x \equiv 10 \pmod{23}$

On obtient:

$$\begin{aligned} x &\equiv 218 \pmod{7^3} \\ x &\equiv 10 \pmod{23} \end{aligned}$$

Le résultat final est $x \equiv 4334 \pmod{7889}$. Cette valeur de x vérifie l'équation $E : y^2 = x^3 + 1001x + 75 \pmod{7919}$ et est donc une solution de l'ECDLP .

La complexité

L'algorithme de Pohlig-Hellman a une complexité temporelle pire des cas de $O(\sqrt{n})$ pour un groupe d'ordre n. Ce qui est intéressant, c'est qu'il est beaucoup plus efficace lorsque l'ordre est B-lisse(friable). Un ordre B-lisse (friable) signifie que l'ordre a des facteurs premiers qui sont inférieurs à B.
Ensuite, si la factorisation en nombres premiers de l'ordre n est $\prod_i p_i^{e_i}$, la complexité sera :

$$O(\sum_i e_i(\log n + \sqrt{p_i}))$$

Il s'avère que la meilleure attaque générale connue sur ECDLP est une combinaison de l'algorithme de Pohlig-Hellman et de l'algorithme rho de Pollard. L'algorithme rho de Pollard est un algorithme basé sur la factorisation d'entiers, et est vraiment efficace pour

un nombre composé ayant un petit facteur premier.
La combinaison de ces algorithmes a un temps d'exécution entièrement exponentiel de $O(\sqrt{p})$ où p est le plus grand facteur premier de l'ordre n.
Il est donc vraiment important de choisir les paramètres de la courbe elliptique de sorte que n soit divisible par un grand nombre premier p, ce qui rend extrêmement difficile et impossible de terminer le calcul en un temps raisonnable.

4.3 L'attaque sur la fonction de hachage

Définition .2. Une fonction de hachage (cryptographique) H est une fonction qui mappe des chaînes de bits de longueurs arbitraires en chaînes de bits de longueur fixe t telle que :

- H peut être calculée efficacement.
- (Résistance à la pré-image) Pour pratiquement tous les $y \in 0, 1^t$, il est computionnellement difficile de trouver une chaîne de bits x telle que $H(x) = y$.
- (Résistance aux collisions) Il est computionnellement difficile de trouver des chaînes de bits distinctes x_1 et x_2 telles que $H(x_1) = H(x_2)$.

Les exigences de sécurité de SHA-1. Ce qui suit explique comment des attaques sur ECDSA peuvent être lancées avec succès si SHA-1 n'est pas résistant à la pré-image ou aux collisions.

1. Si SHA-1 n'est pas résistant à la pré-image, alors un adversaire E peut être en mesure de forger les signatures de A comme suit. E sélectionne un entier arbitraire l et calcule r comme la coordonnée x de $Q + lG$ réduite modulo n. E fixe $s = r$ et calcule $e = rl \mod n$. Si E peut trouver un message m tel que $e = SHA - 1(m)$, alors (r, s) est une signature valide pour m.
2. Si SHA-1 n'est pas résistant aux collisions, alors une entité A peut être en mesure de répudier des signatures comme suit. A génère d'abord deux messages m et m' tels que $SHA - 1(m) = SHA - 1(m')$; un tel paire de messages est appelée une collision pour SHA-1. Elle signe ensuite m et prétend plus tard avoir signé m (notez que chaque signature pour m est également une signature pour m').

Une fonction de hachage de t bits est dite avoir une *sécurité idéale* si :

1. Étant donnée une sortie de hachage, produire une pré-image nécessite environ 2^t opérations.
2. Produire une collision nécessite environ $2^{t/2}$ opérations.

Dans Bitcoin, la courbe de Koblitz $secp256k1$ est combinée avec la fonction de hachage SHA256 dans le processus de signature ECDSA. SHA-256 fait partie de la famille SHA2, standardisée en 2001 par l'Institut national des normes et de la technologie (NIST). La famille SHA-2 restera déployée à l'avenir même en présence de SHA3. SHA-256 est étroitement lié car il utilise des algorithmes très similaires, basé sur les mêmes opérations sur les octets. Il diffère seulement par la longueur des bits en entrée et produise des sorties de 256 bits.

4.4 L'attaque de MOV

Soit G un groupe de courbes elliptiques sur un corps fini K, et soit n un entier premier tel que n soit le plus petit entier tel que $nP = 0$ pour tout P dans G. Le problème du logarithme discret dans G consiste à trouver, étant donné P et Q dans G, un entier m tel que $mP = Q$.

Le groupe multiplicatif d'un corps fini K est noté K^*, et le problème du logarithme discret dans K^* consiste à trouver, étant donné un élément g et un élément h de K^*, un entier x tel que $g^x = h$.

L'attaque MOV consiste à trouver une courbe elliptique E sur un corps fini $\mathbb{F}_q$, avec q premier, telle que le groupe de points $E(\mathbb{F}_q)$ ait un sous-groupe cyclique G de cardinal n, et que la courbe soit telle que les polynômes caractéristiques de l'endomorphisme de Frobenius de E et du corps fini $\mathbb{F}_q$ soient égaux. Cette égalité permet de construire un morphisme de groupes entre $E(\mathbb{F}_q)$ et $(\mathbb{F}_q^*)^2$, qui permet de transformer un problème de logarithme discret dans $E(\mathbb{F}_q)$ en un problème de logarithme discret dans $(\mathbb{F}_q^*)^2$. Puisque le calcul d'indice est possible dans $(\mathbb{F}_q^*)^2$, il peut devenir plus facile de résoudre le DLP et donc de résoudre le ECDLP.

4.4.1 Préliminaires

application bilinéaire

Soient: U, V, W des espaces vectoriels $\{u, u_1, u_2 \in U\}, \{v, v_1, v_2 \in V\}$,α est un scalaire

$$f_2 : U \times V \to W \ (f_2 \text{ est une application de } U \times V \text{ à } W)$$

f_2 est une application bilinéaire si :

$$f_2(u_1 + u_2, v) = f_2(u_1, v) + f_2(u_2, v)$$

$$f_2(u, v_1 + v_2) = f_2(u, v_1) + f_2(u, v_2)$$

$$f_2(\alpha u, v) = \alpha f_2(u, v) = f_2(u, \alpha v)$$

Cela signifie que l'application est linéaire en u si v est fixé et est linéaire en v si u est fixé.

Degré de plongement

Soit E une courbe elliptique définie sur un corps premier $\mathbb{F}_q$. Soit P un point d'ordre m, où m est un nombre premier et est premier avec q.

Si k est le plus petit entier positif tel que

$$q^k \equiv 1 \pmod{m}$$

alors k est appelé le degré de plongement de la courbe $E(\mathbb{F}_q)$ par rapport à m.

Points de torsion et groupe de torsion

Définition .3. Soit E une courbe elliptique définie sur un corps premier $\mathbb{F}_q$. Un point $P \in E(\mathbb{F}_p)$ tel que $mP = O$ est appelé un point de torsion de m.

Définition .4. Le sous-groupe de tous les points de torsion de m dans $E(\mathbb{F}_p)$ est appelé le sous-groupe de torsion de m de $\mathbb{F}_p$ et est représenté par $E(\mathbb{F}_p)[m] = P \in E : mP = O$.

Étant donné que l'extension du corps $\mathbb{F}_{p^c}$ est plus grand que le corps de base $\mathbb{F}_p$, il est probable que le groupe de torsion de m de la courbe sur l'extension du corps soit plus grand que le groupe de torsion de la courbe sur le crps de base. Nous obtenons le plus grand groupe de torsion de m lorsque c est égal au degré d'incorporation de la courbe à m - c'est-à-dire que passer à une extension de corps plus grande que $\mathbb{F}_{p^k}$ n'ajoute pas de points de torsion supplémentaires. Ainsi, $E(\mathbb{F}_{p^k})[m]$ est appelé le groupe complet de torsion de m (où k est le degré d'incorporation de la courbe par rapport à m).

Le groupe de torsion complet a plusieurs sous-groupes, dont deux sont utilisés dans le couplage de Weil :

- G_1 - tous les points de ce sous-groupe sont dans la courbe sur le champ de base, c'est-à-dire qu'ils sont dans $E(\mathbb{F}_p)$.
- G_2 - tous les points de ce sous-groupe sont dans $E(\mathbb{F}_{p^k})$ sans aucun d'entre eux étant dans $E(\mathbb{F}_q)$. Il existe plusieurs de ces sous-groupes, peu importe lequel est choisi.

4.4.2 Accouplement de Weil

Puisque k est le degré de plongement de la courbe par rapport à m,

$$q^k \equiv 1 \bmod m$$

Cela peut s'écrire comme

$$q^k = mx + 1$$
$$q^k - 1 = mx$$

tel que: divise $q^k - 1$
On considére le corps d'extension $\mathbb{F}_{q^k}$. Le groupe multiplicatif de ce corps d'extension, c'est-à-dire $\mathbb{F}_{p^k}$ exclut l'élément 0 et donc son ordre est $q^k - 1$. Puisque m divise l'ordre du groupe multiplicatif, selon le théorème fondamental des groupes cycliques, il possède un sous-groupe unique G_T d'ordre m.
Lorsque la contrainte supplémentaire $m \nmid (q-1)$ est satisfaite, l'accouplement de Weil, qui est une application de $G_1 \times G_2$ vers le groupe multiplicatif G_T, peut être construit.

$$e_m : G_1 \times G_2 \mapsto G_T$$

e_m prend en entrée une paire de $m-$points de torsion , A et B, où $A \in G_1$ et $B \in G_2$. L'application donne en sortie une m-ième racine de l'unité $e_m(A, B)$. La sortie de l'accouplement de Weil, lorsqu'elle est élevée à la puissance m, donne toujours 1, c'est-à-dire $e_m(A, B)^m = 1$.
On note encore une fois que G_1 et G_2 sont des groupes de courbes elliptiques tandis que G_T est

un groupe multiplicatif.

Proposition
Voici les propriétés de l'appariement de Weil :

- **Bilinéarité :** Le côté droit de la bilinéarité de l'accouplement de Weil est multiplicatif plutôt que la bilinéarité additive que nous avons vue précédemment avec les applications bilinéaires des espaces vectoriels.

$$e_m(A_1 + A_2, B) = e_m(A_1, B) \cdot e_m(A_2, B)$$
$$e_m(A, B_1 + B_2) = e_m(A, B_1) \cdot e_m(A, B_2)$$

  Pour un scalaire α,

$$e_m(\alpha A, B) = e_m(A, \alpha B) = e_m(A, B)^\alpha$$

- **Identité :** $e_m(A, B) = 1$ pour tout $A \in E[m]$.
- **Alternance :** $e_{\mathrm{m}}(A, B) = e_{\mathrm{m}}(A, B)^{-1}$ pour tout $A, B \in E[m]$.
- **Non-Dégénérescence :** $e_{\mathrm{m}}(A, O) = 1$ pour tout $A \in E[m]$. Si $e_m(A, B) = 1$ pour tout les $B \in E[m]$, alors cela signifie que $A = O$.

4.4.3 l'attaque MOV :

On considére une courbe elliptique $\boldsymbol{E}$ sur le corps $\mathbb{F}_q$. Nous avons deux points donnés P et Q, tous deux d'ordre premier, tels que $Q = rP$. Nous devons trouver r. Il s'agit du problème du logarithme discret sur la courbe elliptique. Étant donné que P est un point de torsion m, c'est un point sur la courbe sur le corps de base F_p, P, $Q \in G_1$. Les étapes :

1. Calculer l'ordre de la courbe elliptique sur le corps d'extension, c'est-à-dire $n = \#E(\mathbb{F}_q)$. Puisque le groupe de torsion m de $E(\mathbb{F}_p)$ est un sous-groupe de $E(\mathbb{F}_q)$, m divise $\boldsymbol{n}$ (Théorème de Lagrange).
2. Choisir un point aléatoire $T \in E(\mathbb{F}_p)$ tel que $T \neq O$ et $T \in E(\mathbb{F}_p)$.
3. Calculer $S = rP - (2T)$. Si $S = O$, alors revenir à l'étape 2 et choisir un autre point aléatoire T. Si ce n'est pas le cas, alors c'est un point d'ordre m comme indiqué ci-dessous :

$$S = \left(\frac{x}{n}\right) T$$
$$mS - nT$$

   Soit t l'ordre de T. Par le théorème de Lagrange, t divise l'ordre de $E(\mathbb{F}_p)$, c'est-à-dire t divise n. Donc, n peut s'écrire comme $\mathbf{n} = dt$ pour un certain $\mathbf{d}$.

$$\mathrm{mS} = \mathrm{dtT}$$
$$mS = 0$$

   Ainsi, l'ordre de S est m.

4.

$$P, rP \in G_1$$

$$, etS \in G_2$$

On calcule les 2 valeurs d'accouplement de Weil :

$$u = e_{\mathrm{m}}(P, S)$$

$$v = e_{\mathrm{m}}\left(rP, S\right)$$

$$u, v \in \mathbb{F}_{q^k}$$

Étant donné que l'accouplement de Weil est bilinéaire et que r est un scalaire,

$$v = e_m(P, S)^r$$

Puisque $u = e_m(P, S)$, on obtient

$$v = u^r$$

Il s'agit du problème du logarithme discret (DLP) dans le groupe multiplicatif de $\mathbb{F}_{p^k}$. Ainsi, on transforme le problème ECDLP $Q = rP$ en le DLP $u = v^r$.

5. Si p^k n'est pas trop grand, alors $u = v^r$ peut être résolu en utilisant le calcul d'indice et r peut être trouvé. Nous avons donc résolu $Q = rP$.

Si le degré de plongement k est très grand, la transformation du problème ECDLP sur $E(\mathbb{F}_q)$ en un DLP sur $\mathbb{F}_{q^k}$ ne sera pas utile. Cependant, si le degré de plongement est suffisamment petit, le DLP peut devenir considérablement plus facile.

Par exemple, une courbe avec un q de 256 bits offre généralement 128 bits de sécurité. Mais si elle a un degré de plongement de 2, alors nous pouvons mapper le logarithme discret sur le corps $\mathbb{F}_{q^2}$ qui n'offre que 60 bits de sécurité et qui peut être cassé par le calcul d'indice.

Mesures d'atténuation
Pour s'assurer qu'une courbe elliptique E définie sur $\mathbb{F}_q$ est immunisée contre l'attaque MOV, il suffit de vérifier que m, l'ordre du point de base $P \in E(\mathbb{F}_q)$, ne divise pas $q^k - 1$ pour tous les petits k pour lesquels le DLP dans $\mathbb{F}_{q^k}$ est considéré comme gérable. Si l'ordre de P, c'est-à-dire $m > 2^{160}$, alors il suffit de vérifier cette condition pour tous les $k \in [1, 20]$.

5 Etude d'une Attaque :

5.1 Nonce et sécurité de l'ECDSA :

5.1.1 Introduction

L'utilisation de signatures ECDSA est devenue courante dans de nombreux systèmes de sécurité. Cependant, la sécurité de ces systèmes dépend en grande partie de la génération aléatoire de

nonces pour chaque signature. Les nonces sont des nombres aléatoires utilisés pour produire la signature ECDSA, et leur valeur doit être choisie de manière uniforme aléatoire sur un intervalle spécifique. Si les nonces sont mal générés, des attaques sont possibles.

L'attaque de nonce est une technique courante utilisée pour attaquer les signatures ECDSA en exploitant les faiblesses dans la génération de nonces. Plusieurs techniques d'attaque de nonce ont été développées ces dernières années, y compris l'utilisation de PRNG faibles ou de relations de récurrence pour générer des nonces prévisibles.

Dans cette partie de notre rapport, nous allons étudier une nouvelle attaque de nonce décrite dans l'article [1], qui permet d'extraire la clé de signature originale à partir d'une petite collection de signatures ECDSA générées avec des PRNG faibles. Cette attaque utilise des relations de récurrence de degré élevé et inconnu (avec des coefficients inconnus), qui peuvent être attaquées avec quelques signatures et en temps négligeable, sous des conditions appropriées sur l'ordre du modulo du PRNG. Cette attaque est capable de s'attaquer aux relations linéaires, quadratiques, cubiques ainsi qu'aux relations de degré arbitraire.

À notre connaissance, c'est la première attaque connue exploitant des relations algébriques génériques et de haut degré entre les nonces, qui ne nécessite pas d'hypothèses sur la valeur des bits individuels ou des séquences de bits (par exemple, les préfixes et les suffixes). Nous allons examiner cette attaque en détail et discuter des mesures de sécurité qui peuvent être prises pour prévenir ou réduire les risques d'attaques de nonce sur les systèmes ECDSA.

5.1.2 Préliminaire :

Dans cette partie, nous exposons les notions préliminaires essentielles qui seront employées tout au long de cette étude. Par la suite, nous utilisons "ssi" pour "si et seulement si", DLP comme "problème de logarithme discret" et ECDLP pour "problème de logarithme discret de courbe elliptique". Un algorithme ou une procédure fait référence à une série uniforme de circuits ayant une profondeur et une largeur dépendant polynomialement de l'indice de la série. Plus précisément : $F = \{F_n\}_{n\in\mathbb{N}}$ constitue un algorithme ssi il existe un nombre entier $\bar{n}$, des polynômes τ, w, d et une machine de Turing M_F qui, avec n en entrée, génère une représentation de F_n en un temps maximal de $\tau(n)$ pour tout $n > \bar{n}$. Ici, F_n représente un circuit booléen dont la largeur est au plus $w(n)$ et la profondeur est au plus $d(n)$. Si un algorithme A est déterministe, nous représentons sa sortie y pour une entrée x par $y := A(x)$, tandis que s'il est aléatoire, nous utilisons $y \leftarrow A(x)$. Nous emploierons aussi $x \xleftarrow{D} X$ pour signifier qu'un élément x est prélevé à partir d'une distribution D sur un ensemble X (ou simplement $x \leftarrow D$ lorsque l'ensemble d'image est implicite) ; ou nous écrirons $x \xleftarrow{\$} X$ si x est tiré uniformément au hasard d'un ensemble X. Nous qualifierons de négligeable (et noterons $negl(n)$) une fonction qui augmente plus lentement que n'importe quelle fonction polynomiale inverse en n, et d'écrasante une fonction qui est égale à 1 moins une fonction négligeable. Enfin, $a \parallel b$ indique la concaténation de chaînes de caractères.

5.1.3 Primitives cryptographiques et courbes elliptiques :

La cryptographie sur courbes elliptiques est largement utilisée pour sécuriser les communications, les signatures numériques et les protocoles de chiffrement. Dans cette section, nous présentons les bases des courbes elliptiques et leurs primitives cryptographiques, en nous appuyant sur l'article que vous avez partagé.

Une courbe elliptique de Weierstrass E définie sur un corps fini F_q est un ensemble de points $P = (x, y)$ où x et y appartiennent à F_q et satisfont une certaine équation de courbe E, ainsi que le point à l'infini, noté O :

Si $q = p$, $p > 3$ est un nombre premier, chaque point $P = (x, y)$, $P \neq O$, doit satisfaire l'équation suivante dans F_p :

$$E : y^2 = x^3 + ax + b \quad (1) \tag{6}$$

où les paramètres a, $b \in F_p$ sont tels que le discriminant $\Delta = 4a^3 + 27b^2$ n'est pas égal à zéro dans F_p.

Si $q = 2^m$ avec m un nombre premier, chaque point $P = (x, y)$, $P \neq O$, doit satisfaire l'équation suivante dans F_{2^m} :

$$E : y^2 + xy = x^3 + ax^2 + b \quad (2) \tag{7}$$

où le paramètre $b \neq 0$ dans F_{2^m}.

La loi de groupe des courbes elliptiques est définie par l'addition de deux points sur la courbe :

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \tag{8}$$

$$y_3 = \frac{y_2 - y_1}{x_2 - x_1} \cdot (x_1 - x_3) - y_1 \tag{9}$$

Le doublement d'un point est obtenu de manière similaire :

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1^2}\right)^2 - 2x_1 \tag{10}$$

$$y_3 = \frac{3x_1^2 + a}{2y_1} \cdot (x_1 - x_3) - y_1 \tag{11}$$

Pour construire des schémas cryptographiques utilisant des courbes elliptiques, des paramètres de domaine $D = \{q, a, b, G, n, h\}$ doivent être définis, comprenant les éléments suivants :

- Un ordre de champ q

- Une courbe elliptique définie sur F_q par deux coefficients a, $b \in F_q \times F_q$
- Un point de base G, défini par ses coordonnées affines $(x_G, y_G) \in F_q \times F_q$ qui est un point de $E(F_q)$. G a un ordre premier.
- L'ordre n de G
- Le cofacteur $h = E(F_q)/n$

.

L'opération appelée multiplication scalaire-point est construite sur l'opération de groupe d'addition de points de courbes elliptiques :

$$R = [k]P \tag{12}$$

où k est un entier et P un point de $E(F_q)$.

Cette opération est directement liée à l'ECDLP et assure la sécurité dans tous les schémas cryptographiques basés sur ECC. Son temps d'exécution et sa sensibilité aux attaques physiques sont des préoccupations majeures lorsque l'ECC est mise en œuvre en HW et/ou SW. Plusieurs algorithmes réguliers sont disponibles pour exécuter en toute sécurité la multiplication scalaire-point.

5.1.4 Observation générale sur la réutilisation de nonce dans ECDSA

Supposons que N signatures soient générées en utilisant l'algorithme ECDSA en utilisant la même clé privée d ; désignons les paramètres de la i-ème signature comme (k_i, h_i, r_i, s_i) où k_i désigne le nonce, h_i le hash du message, r_i la première moitié de la signature et s_i la seconde moitié de la signature. Toutes les équations dans le reste de cette étude sont destinées modulo n, où n est l'ordre du point générateur de la courbe G. Pour chaque signature générée, nous pouvons écrire :

$$k_i = \frac{h_i}{s_i} + \frac{r_i}{s_i}\ddot{O}d \tag{13}$$

En d'autres termes, la deuxième moitié de chaque signature nous donne une relation linéaire entre le nonce utilisé et la clé privée, car la valeur du hash du message et les deux moitiés de chaque signature sont publiques et connues. Nous pensons que ceci est, en soi, une observation intéressante. Comme la première moitié de chaque signature est la coordonnée x du point $R_i = [k_i]G$, il est simple d'utiliser l'équation de la courbe pour obtenir le point R_i (à un signe près de sa coordonnée y, ici nous pouvons supposer que nous prenons la plus petite valeur de y). Obtenir k_i à partir de R_i signifierait résoudre le problème du logarithme discret sur la courbe ; mais en effet, en utilisant les deuxièmes moitiés des signatures, nous pouvons écrire une relation entre les valeurs des nonces. Prenant l'exemple de deux signatures, il est facile de vérifier que nous pouvons réécrire (6) comme suit :

$$k_1 = \frac{r_1 s_0}{r_0 s_1} k_0 + \frac{(h_1 r_0 - h_0 r_1)}{r_0 s_1} \tag{14}$$

et, en multipliant par le point générateur G, nous obtenons (en regroupant les valeurs connues en a et b) :

$$R_1 = [a]R_0 + [b]G \tag{15}$$

ce qui est une relation non triviale que l'on ne devrait pas pouvoir écrire en connaissant seulement les engagements de nonce (c'est-à-dire les points R_i).

Cela montre que, étant donné l'ensemble des N points R_i, nous ne sommes pas confrontés à N instances du problème du logarithme discret, mais seulement à une instance, car nous savons comment toutes ces valeurs de nonce sont liées, par l'intermédiaire de la clé privée d au moyen des valeurs de s_i. C'est, après tout, la raison derrière le fait bien connu que la connaissance d'une valeur de nonce ECDSA équivaut à la connaissance de la clé privée, et également à la connaissance de toutes les valeurs de nonce jamais utilisées pour générer des signatures avec cette clé privée particulière. Lorsque les nonces utilisés pour N signatures obéissent à une équation polynomiale multivariée, nous pouvons réécrire l'équation en un polynôme univarié impliquant uniquement la clé privée et les valeurs publiques en substituant les relations ci-dessus. Par exemple, si nous connaissons les coefficients a_i et les exposants e_i tels que :

$$a_0 k_0^{e_0} + a_1 k_1^{e_1} + a_2 k_2^{e_2} + \cdots + a_n = 0$$

alors nous pouvons la réécrire comme suit :

$$a_0 \left(\frac{h_0}{s_0} + \frac{r_0}{s_0} d\right)^{e_0} + a_1 \left(\frac{h_1}{s_1} + \frac{r_1}{s_1} d\right)^{e_1} + a_2 \left(\frac{h_2}{s_2} + \frac{r_2}{s_2} d\right)^{e_2} + \cdots + a_N = 0 \tag{16}$$

Maintenant, il s'agit d'un polynôme de degré $\max_i(e_i)$ dans l'inconnue d. Des algorithmes existent pour rapidement développer et trouver les racines de tels polynômes sur des corps finis (par exemple, l'algorithme de Berlekamp) et sont souvent implémentés dans des logiciels de calcul formel librement disponibles, tels que SageMath, par exemple ; la clé privée d apparaîtra toujours parmi les racines du polynôme. Notez que des relations plus complexes où les nonces sont multipliés ensemble fonctionneront également, par exemple,

$$a_0 k_0^{e_{00}} k_1^{e_{01}} \cdots k_{N-1}^{e_{0(N-1)}} + a_1 k_0^{e_{10}} k_1^{e_{11}} \cdots k_{N-1}^{e_{1(N-1)}} + \cdots + a_N = 0$$

peut également être réécrite comme précédemment et mènera également à un polynôme univarié en d (avec potentiellement un degré plus élevé). La question naturelle qui se pose est de savoir comment trouver une telle relation entre les nonces ; s'ils sont choisis de manière correctement aléatoire, cela n'est généralement pas possible. Cependant, dans certains cas, il peut être possible d'écrire une telle relation, comme nous le verrons dans les sections suivantes.

5.1.5 Nonces générés avec une équation de récurrence (inconnue) :

Considérons le cas où le signataire génère N signatures ECDSA en choisissant le premier nonce k_0 aléatoirement et tous les nonces suivants $(N-1)$ en utilisant une relation de récurrence de degré $(N-3)$ avec $N-2$ coefficients inconnus. Nous pouvons considérer cela comme le cas général

d'un générateur de nombres aléatoires polynomiaux avec des coefficients secrets a_i, initialisé par l'état initial k_0. C'est-à-dire, nous pouvons écrire :

$$k_1 = a_{N-3}k_0^{N-3} + a_{N-4}k_0^{N-4} + \cdots + a_1k_0 + a_0 \tag{17}$$
$$k_2 = a_{N-3}k_1^{N-3} + a_{N-4}k_1^{N-4} + \cdots + a_1k_1 + a_0 \tag{18}$$
$$k_3 = a_{N-3}k_2^{N-3} + a_{N-4}k_2^{N-4} + \cdots + a_1k_2 + a_0 \tag{19}$$
$$\cdots \tag{20}$$
$$k_{N-1} = a_{N-3}k_{N-2}^{N-3} + a_{N-4}k_{N-2}^{N-4} + \cdots + a_1k_{N-2} + a_0 \tag{21}$$

Notre objectif est d'écrire un polynôme impliquant uniquement les nonces et d'éliminer les coefficients de récurrence inconnus a_i, de manière à pouvoir ensuite réécrire l'équation avec uniquement la clé privée. Examinons d'abord les cas simples, puis procédons avec un algorithme récursif.

Pour $N = 4$, nous examinons essentiellement le cas simple d'un générateur congruentiel linéaire (LCG) avec un multiplicateur inconnu a_1, un incrément a_0 et un état initial k_0, c'est-à-dire :

$$k_1 = a_1k_0 + a_0 \tag{22}$$
$$k_2 = a_1k_1 + a_0 \tag{23}$$
$$k_3 = a_1k_2 + a_0 \tag{24}$$

Pour éliminer le coefficient a_0, nous soustrayons la deuxième équation de la première et la troisième de la deuxième :

$$k_{1,2} = a_1k_{0,1} \tag{25}$$
$$k_{2,3} = a_1k_{1,2} \tag{26}$$

Où $k_{i,j} = k_i - k_j$. En laissant a_1 des deux côtés droits, nous avons :

$$\frac{k_{1,2}}{k_{0,1}} = a_1 \tag{27}$$
$$\frac{k_{2,3}}{k_{1,2}} = a_1 \tag{28}$$

et en soustrayant à nouveau la deuxième équation de la première, on obtient finalement :

$$\frac{k_{1,2}}{k_{0,1}} - \frac{k_{2,3}}{k_{1,2}} = 0$$

$$k_{1,2}^2 - k_{2,3}k_{0,1} = 0 \quad (18)$$

Cela peut être facilement réécrit comme un polynôme du second degré en d avec des coefficients connus en remplaçant toutes les occurrences de ki,j par :

$$k_{i,j} = \left(\frac{r_i}{s_i} - \frac{r_j}{s_j}\right)d + \frac{h_i}{s_i} - \frac{h_j}{s_j} \quad (19)$$

La clé privée peut alors être facilement récupérée comme l'une des racines de ce polynôme.

Cela signifie que, bien que le signataire utilise un générateur linéaire congruentiel (LCG) modulo n avec des coefficients secrets et un état initial, nous sommes en mesure de récupérer sa clé privée en observant seulement quatre signatures, en un temps négligeable.

Les algorithmes de réduction de réseau sont également capables d'obtenir la clé privée dans le cas linéaire, examinons donc maintenant les récurrences de degrés supérieurs.

Pour N = 5, les équations de récurrence sont :

$$\begin{aligned} k_1 &= a_2 k_0^2 + a_1 k_0 + a_0 \\ k_2 &= a_2 k_1^2 + a_1 k_1 + a_0 \\ k_3 &= a_2 k_2^2 + a_1 k_2 + a_0 \\ k_4 &= a_2 k_3^2 + a_1 k_3 + a_0 \end{aligned}$$

et ce que nous obtenons à la fin après une élimination similaire des coefficients est :

$$(k_{1,2}^2 - k_{2,3}k_{0,1})k_{1,3}k_{2,3} - (k_{2,3}^2 - k_{3,4}k_{1,2})k_{0,1}k_{0,2} = 0$$

ce qui conduit à un polynôme de degré 4 en d après la substitution habituelle ; là encore, d est trouvé comme une racine du polynôme. Ceci représente le cas d'un générateur congruentiel quadratique (QCG) modulo n avec des coefficients secrets et un état initial inconnu.

De même, pour un générateur congruentiel cubique (CCG), nous fixons N = 6 et nous obtenons une équation de degré 7 :

$$\begin{aligned} ((k_{1,2}^2 - k_{2,3}k_{0,1})k_{1,3}k_{2,3} - (k_{2,3}^2 - k_{3,4}k_{1,2})k_{0,1}k_{0,2})k_{1,4}k_{2,4}k_{3,4} \\ - ((k_{2,3}^2 - k_{3,4}k_{1,2})k_{2,4}k_{3,4} - (k_{3,4}^2 - k_{4,5}k_{2,3})k_{1,2}k_{1,3})k_{0,1}k_{0,2}k_{0,3} = 0 \end{aligned}$$

En effet, nous pouvons montrer que les polynômes impliquant les nonces générés par (12) peuvent être obtenus à l'aide d'un algorithme récursif qui construit des polynômes en utilisant les différences de nonces $k_{i,j}$; puis, en remplaçant les valeurs par (19), cela conduit à un polynôme en d qui peut être résolu pour obtenir la clé privée.

L'algorithme récursif a été construit par inspection et généralisation des polynômes construits à la main présentés ci-dessus pour $N = 4, 5, 6$ et a été testé pour fonctionner pour N jusqu'à 16. Le degré en d du polynôme pour N signatures est égal à $1 + \sum_{i=1}^{N-3} i$. Le polynôme en termes de $k_{i,j}$ pour N signatures peut être obtenu en appelant la fonction récursive $\texttt{dpoly}(N-4, N-4, 0)$, où la fonction `dpoly` est décrite dans l'algorithme 2.

[1] $i == 0$ $k_{j+1,j+2}^2 - k_{j+2,j+3}k_{j,j+1}$ $left = \texttt{dpoly}(n, i-1, j)$ $m = 1$ to $i+2$ $left = left \cdot k_{j+m,j+i+2}$ $right = \texttt{dpoly}(n, i-1, j+1)$ $m = 1$ to $i+2$ $right = right \cdot k_{j,j+m}$ $(left - right)$

5.1.6 Ajout d'une signature soigneusement conçue à un ensemble existant

Considérons maintenant le cas où le signataire a correctement généré $N-1$ nonces de manière aléatoire, et a ainsi produit un ensemble de $N-1$ signatures. Les considérations de la section précédente ne sont pas valables pour cet ensemble, mais si nous examinons (12), nous pouvons

voir qu'elles décrivent une équation de récurrence assez générique. Le fait que les coefficients soient inconnus et que nous ne fassions aucune hypothèse à leur sujet, conduit à la considération suivante : prenons toutes les équations de (12) sauf la dernière. Nous avons un ensemble de $N-2$ équations reliant les $N-1$ nonces via les $N-2$ coefficients a_i. Par conséquent, pour un ensemble donné de $N-1$ nonces générés de manière aléatoire, on peut toujours obtenir une solution, c'est-à-dire un ensemble de coefficients a_i qui les relie avec une relation de récurrence de degré $N-3$.

En d'autres termes, même pour un ensemble de nonces aléatoires, il existe toujours une équation de récurrence implicite inconnue de la forme (12) qui les lie. Maintenant, si nous complétons l'ensemble de signatures avec une signature supplémentaire, où le nonce est choisi de manière à satisfaire l'équation de récurrence, nous sommes de retour dans le cas analysé dans la section précédente, et nous pouvons simplement obtenir la clé privée du signataire en utilisant la technique expliquée ci-dessus.
Nous pouvons appeler ce nonce supplémentaire le nonce rogue. Notez qu'en changeant l'ordre des premières $N-1$ signatures, on obtient des valeurs différentes pour les coefficients de la relation de récurrence implicite, et donc une différente valeur du nonce rogue. Lorsque la valeur de N a l'ordre de grandeur du nombre de bits de la courbe, il est facile de voir que, étant donné un ensemble de N signatures correctement générées, il est toujours possible de réorganiser les $N-1$ premières signatures pour que la dernière signature corresponde à la relation de récurrence implicite sous-jacente. Cela découle du fait que le nombre de permutations croît avec le factoriel ou approximativement N^N, qui est supérieur à $2N$. Par conséquent, étant donné un ensemble suffisamment grand de signatures, il y aura toujours une réorganisation donnée qui permettra la récupération de la clé privée par le biais de l'attaque proposée ci-dessus. Tout cela n'a qu'un intérêt théorique, car nous ne sommes pas capables ici de concevoir une façon de trouver cette réorganisation plus facilement qu'avec la force brute.

Formellement, étant donné un ensemble de $N-1$ nonces, les coefficients a_i qui les relient peuvent être obtenus en écrivant les relations de récurrence (12) de 1 à $N-2$ sous forme de matrice, et en inversant la matrice $(N-2)\times(N-2)$ pour obtenir :

$$\begin{bmatrix} a_{N-3} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} k_{N-3} & 0 & \cdots & 0 \\ 0 & k_{N-4} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & k_0 \end{bmatrix}^{-1} \begin{bmatrix} k_1 \\ \vdots \\ k_{N-2} \end{bmatrix}$$

Comme nous pouvons le voir, la connaissance des nonces est nécessaire pour les déduire, ce qui signifie que la valeur du nonce rogue est déterminée de manière unique par l'ensemble des $N-1$ nonces précédents et par leur ordre. Comme on peut le voir, la connaissance des nonces est nécessaire pour les déduire, ce qui signifie que la valeur du nonce rogue est déterminée de manière unique par l'ensemble des N-1 nonces précédents et par leur ordre.

5.1.7 Implémentation de l'attaque sur l'ECDSA:

Dans cette section, nous présenterons l'implémentation de l'attaque ECDSA basée sur l'étude que nous avons examinée. Le code a été écrit en Python en utilisant l'extension SageMath pour faciliter la manipulation des polynômes et des opérations algébriques. Le code complet est disponible dans la référence GitHub [1]

- Paramètres globaux et choix de la courbe

```
usedcurve = ecdsa.curves.SECP256k1
g = usedcurve.generator
d = random.randint(1, usedcurve.order - 1)
pubkey = ecdsa.ecdsa.Public_key(g, g * d)
privkey = ecdsa.ecdsa.Private_key(pubkey, d)
```

Dans cet extrait de code, nous choisissons la courbe elliptique SECP256k1, qui est largement utilisée dans les applications de cryptographie telles que Bitcoin. Le générateur 'g' de la courbe est défini et une clé privée 'd' est générée aléatoirement. Ensuite, la clé publique correspondante est calculée en multipliant le générateur par la clé privée.

- Génération de nonces et de signatures à l'aide d'une relation de récurrence :

```
for i in range(N-1):
    new_k = 0
    for j in range(N-2):
        new_k += a[j]*(k[i]**j) % usedcurve.order
    k.append(new_k)
```

Cet extrait de code montre comment les nonces sont générés à l'aide de la relation de récurrence définie précédemment. La liste 'k' contient les nonces, et la boucle for parcourt les indices pour calculer chaque nonce en utilisant les coefficients de la relation de récurrence et les nonces précédents.

- Construction du polynôme cible :

```
def k_ij_poly(i, j):
    hi = Z(h[i])
    hj = Z(h[j])
    s_invi = Z(s_inv[i])
    s_invj = Z(s_inv[j])
    ri = Z(r[i])
    rj = Z(r[j])
    poly = dd*(ri*s_invi - rj*s_invj) + hi*s_invi - hj*s_invj
    return poly
```

Cette fonction `k_ij_poly` prend deux indices i et j et renvoie un polynôme en fonction de la clé privée `dd`. Les valeurs `hi`, `hj`, `s_invi`, `s_invj`, `ri` et `rj` sont extraites à partir des listes correspondantes de hachages, d'inverses de signatures et de paramètres r. Ce polynôme est utilisé pour construire le polynôme cible dans l'étape suivante de l'implémentation.

- Génération de signatures:

```
for i in range(N):
    digest_fnc = hashlib.new("sha256")
    digest_fnc.update(b"recurrence test ")
    digest_fnc.update(i.to_bytes(1, 'big'))
    h.append(digest_fnc.digest())
    if usedcurve.order.bit_length() < 256:
        h[i] = (int.from_bytes(h[i], "big") >> (256 - usedcurve.order.bit_length()))
        % usedcurve.order
    else:
        h[i] = int.from_bytes(h[i], "big") % usedcurve.order
    sgns.append(privkey.sign(h[i], k[i]))
```

Dans cette section du code, les signatures ECDSA sont générées en utilisant les nonces calculés précédemment. Pour chaque nonce, un hachage est créé à l'aide de la fonction de hachage SHA-256. Ensuite, la signature est générée en utilisant la clé privée et le nonce correspondant.

- Récupération des paramètres de signature:

```
for i in range(N):
    s.append(sgns[i].s)
    r.append(sgns[i].r)
    s_inv.append(ecdsa.numbertheory.inverse_mod(s[i], usedcurve.order))
```

Dans cette partie, les paramètres de signature 's' et 'r' sont extraits des objets de signature ECDSA, et les inverses modulaires de 's' sont calculés.

- Recherche de la clé privée parmi les racines du polynôme:

```
d_guesses = poly_target.roots()
for i in d_guesses:
    if i[0] == d:
        print("key found!!!")
```

Après avoir construit le polynôme cible, cette section du code tente de récupérer la clé privée en recherchant les racines du polynôme. Si l'une des racines correspond à la clé privée d'origine, cela signifie que l'attaque a réussi à retrouver la clé privée.

Conclusion : Dans l'ensemble, le code présenté implémente une attaque sur le schéma de signature ECDSA en exploitant la vulnérabilité introduite par l'utilisation de nonces liés par une relation de récurrence. L'objectif principal de cette attaque est de récupérer la clé privée à partir d'un ensemble de signatures ECDSA générées avec des nonces faibles.

Le code commence par définir les paramètres globaux, générer une clé privée aléatoire et choisir une courbe elliptique à utiliser. Ensuite, il génère des nonces en utilisant une relation de récurrence, puis crée des signatures ECDSA en utilisant ces nonces et la clé privée. Le code construit ensuite un polynôme en utilisant les paramètres des signatures générées et cherche les racines de ce polynôme pour tenter de retrouver la clé privée.

Si l'une des racines du polynôme correspond à la clé privée originale, cela signifie que l'attaque a réussi et la clé privée a été récupérée. Dans le cas contraire, l'attaque n'a pas réussi à retrouver la clé privée à partir des signatures données.

En étudiant cette attaque et en implémentant ce code, nous avons pu comprendre l'importance d'utiliser des nonces cryptographiquement sécurisés lors de la génération de signatures ECDSA. L'utilisation de nonces faibles ou prévisibles peut entraîner la divulgation de la clé privée, compromettant ainsi la sécurité du schéma de signature.

6 Paramétres de l'ECDSA et considérations de sécurité

Les paramètres de domaine pour l'ECDSA peuvent être représentés mathématiquement de la manière suivante :

Soit E une courbe elliptique définie sur un corps fini $\mathbb{F}_{\shortparallel}$ de caractéristique p. On choisit un point de base $G \in E(\mathbb{F}_q)$, tel que l'ordre du groupe généré par G soit un nombre premier n.

Les clés privées k utilisées dans ECDSA sont des entiers choisis aléatoirement, et les clés publiques correspondantes K sont calculées en multipliant le point de base G par la clé privée k: $K = kG$.

La sécurité de l'algorithme dépend du choix approprié de la courbe elliptique E et du point de base G, qui doivent être choisis de manière à garantir que la détermination de la clé privée k à partir de la clé publique K soit difficile.
Des restrictions sont placées sur la taille du corps fini $\mathbb{F}_{\shortparallel}$ et sur la représentation des éléments de ce corps. Des restrictions sont également imposées sur la courbe elliptique E et sur l'ordre n du point de base G afin de prévenir certaines attaques connues

6.0.1 Les contraintes sur le corps

L'ordre du corps fini de base est soit $q = p$, un nombre premier impair, soit $q = 2^m$, une puissance de 2. Dans le cas où $q = p$, le corps de base est F_p, c'est-à-dire les entiers modulo p. Dans le cas où $q = 2^m$, le corps fini sous-jacent est $\mathbb{F}_{2^m}$

6.0.2 Les contraintes sur la courbe elliptique

Pour éviter certaines attaques (Pollard's rho et Pohlig-Hellman) il est nécessaire que le nombre de points F_q-rational sur E soit divisible par un nombre premier suffisamment grand n.
ANSI X9.62 impose que $n > 2^{160}$. Après avoir fixé un corps de base $\mathbb{F}_{\text{ıı}}$, n doit être choisi le plus grand possible, c'est-à-dire que $n \approx q$, de sorte que $\#E(\mathbb{F}_{\text{ıı}})$ soit presque premier. Dans le reste, nous supposerons que $n > 2^{160}$ et que $n > 4\sqrt{q}$.
Le co-facteur est défini comme étant $h = \#E(\mathbb{F}_q)/n$.

D'autres précautions doivent être prises lors de la sélection de la courbe elliptique:
la courbe doit être non supersingulière (c'est-à-dire que p ne doit pas diviser $(q + 1 - \#E(\mathbb{F}_q)$. Plus généralement, on doit vérifier que n ne divise pas $q^k - 1$ pour tout $1 \leq k \leq C$, où C est suffisamment grand pour qu'il soit computationnellement impossible de trouver des logarithmes discrets dans $\mathbb{F}_{q^C}$ ($C = 20$ suffit en pratique).
Enfin, la courbe ne doit pas être $\mathbb{F}_q$-anormale (c'est-à-dire $\#E(\mathbb{F}_q) \neq q$).

Une manière prudente de se protéger contre ces attaques et d'autres attaques contre des classes spéciales de courbes qui pourraient être découvertes à l'avenir est de sélectionner la courbe elliptique E de manière aléatoire sous réserve que $\#E(\mathbb{F}_q)$ soit divisible par un grand nombre premier - la probabilité qu'une courbe aléatoire succombe à ces attaques à usage spécifique est négligeable. Une courbe peut être sélectionnée de manière vérifiable de manière aléatoire en choisissant les coefficients de l'équation elliptique définissant la courbe comme les sorties d'une fonction à sens unique telle que SHA-1 selon une procédure pré-spécifiée.

6.1 Génération d'une courbe elliptique vérifiablement aléatoire

Cette sous-section décrit la méthode utilisée pour générer une courbe elliptique vérifiablement aléatoire. Les paramètres de définition de la courbe elliptique sont définis comme des sorties de la fonction de hachage à sens unique SHA-1 (telle que spécifiée dans FIPS 180-1). La graine d'entrée de SHA-1 sert alors de preuve (sous l'hypothèse que SHA-1 ne peut pas être inversée) que la courbe elliptique a été effectivement générée de manière aléatoire. Cela offre une certaine assurance à l'utilisateur de la courbe elliptique que l'entité qui a généré la courbe elliptique n'a pas intentionnellement construit une courbe "faible" que l'entité pourrait ensuite exploiter pour récupérer les clés privées de l'utilisateur. L'utilisation de cette méthode de génération peut également aider à atténuer les préoccupations concernant la découverte possible à l'avenir de nouvelles et rares classes de courbes elliptiques faibles, car de telles courbes rares ne seraient essentiellement jamais générées.

6.1.1 Le cas $q = p$

La notation suivante est utilisée : $t = \log_2(p)$, $s = \frac{t-1}{160}$, et $v = t - 160s$.

Algorithme 1 : Génération d'une courbe elliptique aléatoire sur $\mathbb{F}_p$.

Entrée : Un corps de p, où p est un nombre premier impair.

Sortie : Une chaîne de bits seedE d'une longueur d'au moins 160 bits et des éléments de corps $a, b \in \mathbb{F}_p$ qui définissent une courbe elliptique E sur $\mathbb{F}_p$.

1. Choisir une chaîne de bits arbitraire seedE d'une longueur $g \leq 160$ bits.
2. Calculer H = SHA-1(seedE), et laisser c_0 désigner la chaîne de bits de longueur v obtenue en prenant les v bits à droite de H.
3. W_0 désigne la chaîne de bits de longueur v obtenue en positionnant le bit à gauche de c_0 à 0. (Cela garantit que $r < p$.)
4. Soit z l'entier dont l'expansion binaire est donnée par la chaîne de bits seedE de longueur g.
5. Pour i allant de 1 à s faire :
   (a) Soit s_i la chaîne de bits g-bit qui est l'expansion binaire de l'entier $(z + i) \bmod 2^g$.
   (b) Calculer $W_i = \text{SHA-1}(s_i)$.
6. Soit W la chaîne de bits obtenue en concaténant W_0, W_1, ..., W_s comme suit : $W = W_0||W_1||...||W_s$.
7. Soit r l'entier dont l'expansion binaire est donnée par W.
8. Si $r = 0$ ou si $4r + 27 \equiv 0 \pmod{p}$, alors revenir à l'étape 1.
9. Choisir arbitrairement les entiers $a, b \in \mathbb{F}_p$, tels que $(a, b) \neq (0, 0)$, et $r \cdot b^2 \equiv a^3 \pmod{p}$. (Par exemple, on peut prendre $a = r$ et $b = r$.)
10. La courbe elliptique choisie sur $\mathbb{F}_p$ est $E : y^2 = x^3 + ax + b$.
11. Retourner (seedE, a, b).

Algorithme 2 : Vérification qu'une courbe elliptique a été générée de manière aléatoire sur $\mathbb{F}_{\text{।}}$.

Entrée : Un corps de p élements (un nombre premier), une chaîne de bits *seedE* de taille $g \geq 160$ bits, et a, $b \in F_p$ des éléments du corps qui définissent une courbe elliptique $E : y^2 = x^3 + ax + b$ sur F_p.

Sortie : Acceptation ou rejet que E a été généré de manière aléatoire en utilisant l'algorithme 1.

1. Calculer $H = SHA - 1(seedE)$, et laisser c_0 désigner la chaîne de bits de longueur v obtenue en prenant les v bits de droite de H.
2. Soit W_0 désigne la chaîne de bits de longueur v obtenue en mettant le bit de gauche de c_0 à 0.

3. Soit z être l'entier dont l'expansion binaire est donnée par la chaîne de bits de g-bit $seedE$.
4. Pour i allant de 1 à s faire :
   (a) Soit s_i être la chaîne de bits de g-bit qui est l'expansion binaire de l'entier $(z + i) \bmod 2^g$.
   (b) Calculer $W_i = SHA - 1(s_i)$.
5. Soit W être la chaîne de bits obtenue en concaténant W_0, W_1, ..., W_s comme suit : $W' = W_0||W_1||\cdots||W_s$.
6. Soit r' être l'entier dont l'expansion binaire est donnée par W'.
7. Si $r' \cdot b^2 \equiv a^3 \pmod p$ alors accepter ; sinon rejeter.

6.1.2 Cas $q = 2^m$

La notation suivante est utilisée : $s = \lfloor (m-1)/160 \rfloor$ et $v = m - 160 \cdot s$.

Algorithme 3 : Génération d'une courbe elliptique aléatoire sur $\mathbb{F}_{2^m}$.

Entrée : Une taille de champ $q = 2^m$.

Sortie : Une chaîne de bits $seedE$ d'une longueur d'au moins 160 bits et des éléments de champ $a, b \in \mathbb{F}_{2^m}$, qui définissent une courbe elliptique E sur $\mathbb{F}_{2^m}$.

1. Choisir une chaîne de bits arbitraire $seedE$ de longueur $g \geq 160$ bits.
2. Calculer $H = SHA - 1(seedE)$, et soit b_0 la chaîne de bits de longueur v obtenue en prenant les v bits de droite de H.
3. Soit z l'entier dont l'expansion binaire est donnée par la chaîne de bits de g-bit $seedE$.
4. Pour i allant de 1 à s faire :
   (a) Soit s_i la chaîne de bits de g-bit qui est l'expansion binaire de l'entier $(z+i) \bmod 2^g$.
   (b) Calculer $b_i = SHA - 1(s_i)$.
5. Soit b l'élément de corps obtenu en concaténant $b_0, b_1, ..., b_s$ comme suit : $b = b_0||b_1||\cdots||b_s$.
6. Si $b = 0$, alors retourner à l'étape 1.
7. Soit a être un élément arbitraire de $\mathbb{F}_{2^m}$.
8. La courbe elliptique choisie sur $\mathbb{F}_{2^m}$ est $E : y^2 + xy = x^3 + ax^2 + b$.
9. Retourner $(seedE, a, b)$.

Algorithme 4: Vérification qu'une courbe elliptique a été générée aléatoirement sur $\mathbb{F}_{2^m}$.

Entrée : Une taille de champ $q = 2^m$, une chaîne de bits $seedE$ de longueur $g \geq 160$ bits et des éléments de champ $a, b \in \mathbb{F}_{2^m}$ qui définissent une courbe elliptique $E : y^2 + xy = x^3 + ax^2 + b$

sur $\mathbb{F}_{2^m}$.

Sortie : Acceptation ou rejet de la proposition que E a été générée aléatoirement en utilisant l'algorithme 3.

1. Calculer $H = \text{SHA-1}(seedE)$, et soit b_0 la chaîne de bits de longueur v obtenue en prenant les v bits de droite de H.
2. Soit z l'entier dont l'expansion binaire est donnée par la chaîne de bits de g-bit $seedE$.
3. Pour i allant de 1 à s faire :
   (a) Soit s_i être la chaîne de bits de g-bit qui est l'expansion binaire de l'entier $(z + i) \bmod 2^g$.
   (b) Calculer $b_i = \text{SHA-1}(s_i)$.
4. Soit b' l'élément de champ obtenu en concaténant $b_0, b_1, \ldots, b_s$ comme suit: $b' = b_0||b_1||\cdots||b_s$. Si $b = b'$ alors accepter ; sinon rejeter.

7 Conclusion

En conclusion de cette étude sur la vulnérabilité de l'algorithme ECDSA (Elliptic Curve Digital Signature Algorithm), plusieurs considérations de sécurité cruciales ont été abordées. Tout d'abord, l'analyse a révélé l'existence d'attaquesgénériques, notamment l'attaque de Pohlig-Hellman, qui exploite la factorisation de l'ordre du groupe de la courbe elliptique pour résoudre le problème du logarithme discret. Dans ce contexte, il est fondamental de sélectionner des courbes elliptiques présentant un ordre de groupe important afin de renforcer la sécurité de l'ECDSA.

Par ailleurs, les attaques visant les fonctions de hachage utilisées dans le processus de signature de l'ECDSA ont été examinées. Ces vulnérabilités peuvent conduire à la découverte de collisions ou à l'exploitation de faiblesses cryptographiques, mettant ainsi en péril la fiabilité des signatures. Par conséquent, l'utilisation de fonctions de hachage robustes et résistantes aux attaques connues s'avère cruciale pour garantir un niveau de sécurité adéquat.

Une attention particulière a également été portée à l'attaque MOV (Menezes-Okamoto-Vanstone) qui exploite l'appariement de Weil sur les courbes elliptiques pour résoudre le problème du logarithme discret. Les résultats de cette étude soulignent la nécessité de privilégier des courbes elliptiques avec un ordre de groupe élevé afin de contrecarrer les attaques basées sur le protocole MOV.

Enfin, l'attaque de nonce a été prise en considération, celle-ci exploitant la réutilisation de valeurs de nonce dans le processus de signature de l'ECDSA. Lorsque les nonces sont réutilisés, des informations sensibles peuvent être divulguées, ce qui compromet la confidentialité des clés privées. Ainsi, pour prévenir cette vulnérabilité, il est primordial de générer des nonces aléatoires et uniques pour chaque signature effectuée.

En conclusion, l'étude des vulnérabilités de l'ECDSA souligne l'importance de se préoccuper des avancées en cryptographie quantique ou post-quantique. L'émergence de l'informatique quantique remet en question les algorithmes traditionnels sur lesquels repose l'ECDSA, en les rendant vulnérables aux attaques quantiques.

Pour faire face à cette menace, la cryptographie quantique ou post-quantique offre des solutions innovantes, basées sur de nouveaux fondements mathématiques et des primitives de chiffrement résistantes aux attaques quantiques. Les protocoles de signature post-quantiques représentent une alternative prometteuse à l'ECDSA, en fournissant une sécurité renforcée dans un contexte de calcul quantique.

Il est essentiel de suivre attentivement les développements de la cryptographie quantique ou post-quantique, car cela pourrait nécessiter une transition vers de nouvelles normes et protocoles pour garantir la sécurité des communications à long terme.

En somme, l'étude des vulnérabilités de l'ECDSA met en évidence la nécessité d'explorer des solutions avancées en cryptographie quantique ou post-quantique, afin d'assurer une sécurité durable face aux avancées technologiques et aux menaces potentielles des ordinateurs quantiques. La recherche continue dans ce domaine est cruciale pour façonner l'avenir de la cryptographie et préserver la confidentialité et l'intégrité des communications dans un paysage informatique en constante évolution.

Références

1. (2023) de https://eprint.iacr.org/2023/305.pdf

1 Kudelski Security. (n.d.). *ecdsa-polynomial-nonce-recurrence-attack* de https://github.com/kudelskisecurity/ecdsa-polynomial-nonce-recurrence-attack

2. E-ducat.fr. (n.d.). *ECDSA, technologie clé de bitcoin* de https://web.archive.org/web/*/http://www.e-ducat.fr/ecdsa-technologie-cle-de-bitcoin

3. ANSI. (1998). *ANSI X9.62 - The Elliptic Curve Digital Signature Algorithm (ECDSA).* Récupéré le 8 mai 2023, de https://www.pdfdrive.com/ansi-x962-1998-the-elliptic-curve-digital-signature-algorithm-ecdsa-e[liencompletouIDdudocu

4. Inconnu. (n.d.). *Elliptic Curve Digital Signature Algorithm*de https://maaz.ihmc.us/rid=1V1Z7BB5X-ZBQ686-2X4/Elliptic%20Curve%20Digital%20Signature%20Algorithm.pdf

5. National Institute of Standards and Technology. *Secure Hash Standard (SHS).* FIPS Publication 180-1. 1995. Web.

6. Johnson, D., Menezes, A. Vanstone, S (2001). The Elliptic Curve Digital Signature Algorithm (ECDSA) *Springer-Verlag 2001*

7. *Ballet, S. Bonecaze, A. (2011-2012).* Cours de cryptographie avancée *Courbes elliptiques: application à la cryptographie*

8. *https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introducti*

9. *Tun Myat Aung, Ni Ni Hla. A Study of General Attacks on Elliptic Curve Discrete Logarithm Problem over Prime Field and Binary Field* International Journal of Information, Control and Computer Sciences ISSN: 2517-9942 Vol:11, No:11, 2017

10. Martin Lysoe Sommerseth and Haakon Hoeiland Pohlig-Hellman Applied in Elliptic Curve Cryptography December 7, 2015