UNIVERSITE KASDI MERBAH OUARGLA

Faculté Des Sciences Et De La Technologie Et Sciences De La Matière

Département De Mathématique Et Informatique

Mémoire

MASTER ACADEMIQUE

Domaine : Mathématique et Informatique
Filière : Informatique Académique
Spécialité : Informatique Industrielle

Présenté par : Mahammedi Nadjiba

Mahdadi Houda

Thème

Implémentation de benchmark d'opérations crypto basées ECC pour l'étude et comparaison de courbes elliptiques $\mathbb{F}_p$ et $\mathbb{F}_{2^n}$

Soutenu publiquement
le : ../06/2013
Devant le jury :

M. MEFLAH Med Salim MC(A)	Président	UKM Ouargla
M. EUSCHI Salah MC(A)	Encadreur/rapporteur	UKM Ouargla
M. MAHDJOUBE Med Bachir MC(A)	Examinateur	UKM Ouargla

Année Universitaire : 2012 /2013

Dédicaces

Je dédie ce travail ;

A mes très cher mes parents,

qui ont éclairé mon chemin et qui m'ont encouragé

et soutenue tout au long de mes études.

Mes frères et mes sœurs,

Toute ma famille,

Mahdadi et Ben Hadja

A tout mes collègues d'étude,

Ainsi qu'à tous ceux qui me sont chers.

A Nadjiba Mahammedi.

Houda

Dédicaces

Je dédie ce travail :
Ames parents qui ont éclairé mon chemin et qui m'ont encouragé et soutenue tout au long de mes études. Ames frères , Ames sœurs Ames chères amies Gouffrane, Halima, Halima, Chafya, ma binôme MAHDADI Houda, et a toute sa famille,
Enfin je dédie ce travail à mes chers amis et consultant et à tous ceux, qui de près ou de loin ont contribués à la réalisation de mon mémoire et toutes les promotions Informatique Industrielle, Informatique Fondamentale.

Nadjiba

REMERCIEMENTS

Avant tout, nous remercions Dieu de nous avoir donné le courage, la volonté, la patience et la chance de suivre le chemin de la Science.

Nous remercions respectueusement Mr. EUSCHI Salah, maître assistant au département de Mathématique et Informatique de l'université Kasdi Merbah Ouargla pour avoir accepté de diriger ce travail.

Nos sincères remerciements Mr . MEFLAH Mohamed Salim, maître assistant au département de Mathématique et Informatique de l'université Kasdi Merbah Ouargla, pour présider le jury de soutenance.

Nous remercions respectueusement Mr . MAHDJOUBE Med Bachir, maître assistant au département de Mathématique et Informatique de l'université Kasdi Merbah Ouargla, pour les conseils et avoir accepté d'examiner ce mémoire.

Nous remercions respectueusement Mr. GUERBOUSSA Yassine et REGUIAT Miloud pour leurs aides dans l'explication des concepts algébriques des courbes elliptiques et leurs conseils.

Un grand merci encore vivement à nos grandes familles MAHAMMEDI et MAHDADI.

Nous remercions sincèrement les enseignants du département de Mathématique et Informatique, nos très chers amis et collègues.

Enfin, Nos sincères remerciements vont également à tous ceux et celles Qui ont contribué de prés et de loin à la réalisation de ce modeste travail.

Table des matières

Liste des figures

Liste des tables

Abréviation

A

AES Advanced Encryption Standard
ANSI American National Standards Institute
API Application Programmation Interface
A5 Audio 5

B

BC Bouncy Castle

C

CA Certification Authority
CBC Cipher-Block Chaining mode of operation
CFB Cipher Feedback mode of operation
CPU Computer Processing Unite
CRL liste de révocation de certificat
CRT Chinese Remainder Theorem
CSP Cryptographic Service Provider

D

DES Data Encryption Standard
DH Diffie-Hellman
DL Discrete Logarithm
DLP Discrete Logarithm Problem
DSA Digital Signature Algorithm
DSS Digital Signature Standard

E

ECB Electronic Codebook mode of operation
ECC Elliptic Curve Cryptography
ECDH Elliptic Curve Diffie-Hellman
ECDSA Elliptic Curve Digital Signature Algorithm
ECIES Elliptic Curve Integrated Encryption Scheme
EDI Environnement de Développement Intégré

F

FIPS Federal Information Processing Standards

G

GF(p) Corps fini (Galois Field) premier à p éléments
GF(2n) Corps fini (Galois Field) binaire à 2n éléments
GSM Global System for Mobile

H

HMAC Hash-based Message Authentication Code

I

IBM International Business Machines
IDEA Norme internationale de cryptage de données
IEEE Institute of Electrical and Electronics Engineers
IPSec Internet Protocol Security
ISO Organisation internationale de normalisation
IV Initialization Vector

J

JCA Java Cryptography Architecture
JCE Java Cryptography Extension
JRE Java Runtime Environnement
JVM Machine Virtuelle Java

K

KDF Key Derivation Function

M

MAC Message Authentication Code
MD4 Message Digest 4
MD5 Message Digest 6

N

NIST National Institute of Standards and Technology
NSA National Security Agency

P

PBE Password Based Encryption
PKCS Public Key Cryptography Standards
PKI Public-Key Infrastructure
PRNG Random Number Generator

R

RC4 Rivest Cipher 4
RC6 Rivest Cipher 6
RSA Rivest-Shamir-Adleman

S

SEC Standards for Efficient Cryptography
SECG Standards for Efficient Cryptography Group
SET Transaction électronique sécurisée
SHA-1 Secure Hash Algorithm 1
SSL Secure Sockets Layer
SPI Service Provider Interface
S/MIME extension de messagerie Internet multi-usages

T

TLS Transport Layer Security
TDES Triple Data Encryption Standard

W

WEP Wired Equivalent Privacy

X

XOR eXclusive OR

Introduction Générale

Suite au développement rapide ces dernières années des technologies de l'information et la communication et les infrastructures réseaux filaires et sans fil, la cryptographie est devenue, depuis quelques décennies, un véritable enjeu de société. Le développement rapide des réseaux de communication numériques, ainsi que l'augmentation du nombre de ses utilisateurs, ont posé de façon de plus en plus critique le problème de l'échange des clés de chiffrement. Très schématiquement, le principe de la cryptographie classique consiste à échanger des données chiffrées à partir d'un secret (appelé clé) connu uniquement par les parties concernées, le chiffrement comme le déchiffrement nécessitant la clé. Le paradoxe étant que pour pouvoir échanger des informations secrètement, les deux parties doivent déjà partager un secret, le problème d'échange de clés prend une tout autre dimension à l'heure d'internet et de son milliard d'usagers. C'est pour répondre à ce problème qu'a été inventée la cryptographie à clé publique, le principe de la cryptographie à clé publique est d'utiliser non plus une clé pour chiffrer et déchiffrer mais un couple de clés : une clé publique, donc connue de tous, servant au chiffrement, et une clé privée, connue uniquement du destinataire, permettant le déchiffrement. Ainsi si une personne a besoin de se faire envoyer des informations de façon confidentielle, il lui suffit de générer elle même un couple clé publique-clé privée et plus aucun échange de clés n'est alors nécessaire (mais se pose alors le problème de la diffusion de cette clé publique).Si le principe de la cryptographie à clé publique fut proposé en 1975, ce n'est qu'en 1977 que fut présenté le premier protocole effectif : RSA. Principalement basé sur le problème de la factorisation des grands entiers, RSA est encore aujourd'hui le crypto système le plus utilisé en cryptographie. Cependant les nombreux progrès effectués dans le domaine de la factorisation font que la taille des clés RSA augmente plus vite que ne le requiert l'augmentation de la puissance des ordinateurs. C'est l'une des raisons pour lesquelles la cryptographie basée sur les courbes elliptiques (ECC) connaît un tel intérêt depuis son introduction par Miller et Koblitz en 1985.

Les systèmes cryptographiques basés sur les courbes elliptiques permettent d'obtenir un gain en efficacité dans la gestion de clés. En effet, de tels crypto systèmes nécessitent des clés de taille beaucoup plus modeste, par exemple, une clé ECC de 160 bits réalise le même niveau de sécurité qu'une clé RSA 1024 bits, ce qui représente un avantage pour les systèmes utilisant les cartes à puces dont l'espace mémoire est très limité. De plus, les algorithmes de calculs liés aux courbes elliptiques sont plus rapides, et ont donc un débit de générations et d'échanges de clé beaucoup plus important.

L'objectif de notre travail consiste à réaliser des benchmarks pour mesurer les opérations crypto basées ECC, en choisissant des courbes elliptiques appartenant aux différentes familles (courbes PF, BF), nous avons choisi d'utiliser des courbes prédéfinies (courbes nommés) par des standards de sécurité comme NIST et SEC, ces courbes sont vérifiés et validés par ces standards. Ces benchmarks de comparaison nous aident dans le choix des courbes elliptiques les plus appropriées à notre environnement matériel et logiciel utilisé. Nous avons choisi d'utiliser le package crypto de Bouncy Castle sous la plateforme Java J2SE, parce qu'il est open source, riche en algorithmes crypto et implémente des classes implémentant des services crypto basés sur la cryptographie de courbes elliptiques.

Notre mémoire est organisé en quatre chapitres, dans le premier chapitre nous présenterons les concepts fondamentaux de la cryptographie moderne comme ils sont définis par les standards internationaux de la sécurité, nous étudierons, dans le deuxième chapitre nous montrons certains concepts mathématiques relatives aux courbes elliptiques et comment utiliser certains de ces courbes en cryptographie, ensuite nous présenterons dans le troisième chapitre l'environnement pratique de notre travail qui consiste à la plateforme Java avec son architecture de cryptographie JCA/JCE et le provider JCE de BouncyCastle, et enfin nous étudierons l'implémentation de benchmarks d'opérations crypto basées ECC pour l'étude et la comparaison de ces courbes elliptiques.

Chapitre I :

Les Concepts Fondamentaux De La Cryptographie Moderne

I.1 Introduction

Dans la société de l'information d'aujourd'hui, l'usage de la cryptologie s'est banalisé. On le retrouve quotidiennement avec les cartes bleues, téléphones portables, internet ou encore les titres de transport. La cryptologie moderne a pour l'objet l'étude des méthodes qui permettent d'assurer les services d'intégrité, d'authenticité et de confidentialité dans les systèmes d'information et de communication. Elle recouvre aujourd'hui également l'ensemble des procédés informatiques devant résister à des adversaires. La cryptographie moderne est basée sur les mathématiques pour sécuriser l'information.

I.2 Histoire de la cryptographie

Les ordinateurs et le réseau Internet font entrer la cryptologie dans son ère moderne. La grande invention de ces dernières décennies fut la cryptographie à clefs publiques :

1970 : Au début des années 1970, Horst Feistel a mené un projet de recherche à l'IBM Watson Research Lab qui a développé le chiffre Lucifer, qui inspira plus tard le chiffre DES et d'autres chiffres.

1976 : Whitfield Diffie et Martin Hellman publient New Directions in cryptography, introduisant l'idée de cryptographie à clef publique. Ils donnent une solution entièrement nouvelle au problème de l'échange de clés.

Novembre 1976 : DES est un algorithme très répandu à clé privée dérivé du chiffre Lucifer de Feistel (de chez IBM) dans sa version à 64 bits.

Avril 1977 : RSA, Le brevet de cet algorithme appartient à la société américaine RSA Data Security. RSA est un algorithme à clé publique qui sert aussi bien à la cryptographie de documents, qu'à l'authentification.

1978 : L'algorithme RSA est publié dans les Communications de l'ACM.

1985 : Victor Miller et Neal Koblitz utilisent les courbes elliptiques pour la cryptographie [01].

I.3 Définition de la cryptographie

La cryptographie est l'art de rendre inintelligible, de crypter, de coder un message à ceux qui ne sont pas habilités à en prendre connaissance ou est un ensemble des principes, méthodes et techniques dont l'application assure le « crypter » et le « décrypter » des données [03,04](Figure I.1). La signification de « crypter » et « décrypter » est donnée successivement comme suit:

- **Crypter:** brouiller l'information, la rendre "incompréhensible"
- **Décrypter:** rendre le message compréhensible

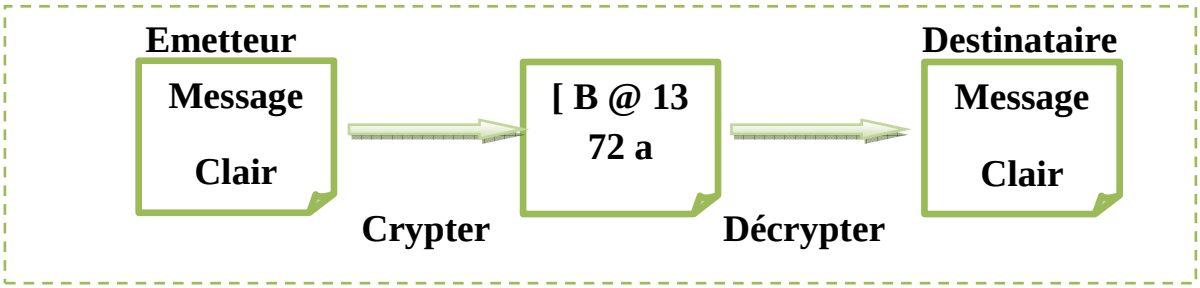


Figure I.1: Chiffrement / déchiffrement

I.4 Objectif de cryptographie

Les principaux objectifs à garantir par l'application de la cryptographie sont**:**

Confidentialité: un mécanisme pour transmettre des données de telle sorte que seul le destinataire autorisé puisse les lire.

Intégrité des données: un mécanisme pour s'assurer que les données reçues n'ont pas été modifiées durant la transmission, frauduleusement ou accidentellement.

Authentification: un mécanisme pour permettre d'authentifier les utilisateurs de façon à limiter l'accès aux données, serveurs et ressources aux seules personnes autorisées (un mot de passe par un nom de login ou un certificat numérique).

Non-répudiation: un mécanisme pour enregistrer un acte ou un engagement d'une personne ou d'une entité de telle sorte que celle-ci ne puisse pas nier avoir accompli cet acte ou pris cet engagement. Ce mécanisme se décompose:

- non-répudiation d'origine l'émetteur ne peut nier avoir écrit le message.
- non-répudiation de réception le receveur ne peut nier avoir reçu le message.
- non-répudiation de transmission l'émetteur du message ne peut nier avoir envoyé le message. [02]

I.5 Techniques de cryptographie

Pour assurer les objectifs de la cryptographie nous pouvons utiliser des algorithmes basés sur des clés. Ces algorithmes sont définis par plusieurs types de cryptographie ou crypto systèmes :

I.5.1 Cryptographie symétrique

Le chiffrement symétrique (aussi appelé chiffrement à clé privée ou à clé secrète) consiste à utiliser la même clé pour le chiffrement et le déchiffrement (Figure I.2). Le chiffrement consiste alors à effectuer une opération entre la clé privée et les données à chiffrer afin de rendre ces dernières inintelligibles. Le déchiffrement consiste à réaliser l'opération inverse c'est-à-dire récupérer le message d'origine à partir du message chiffré en utilisant la clé secrète.

Figure I.3 : Chiffrement symétrique

Les algorithmes symétriques sont plus rapides et facilement implémentés sur le matériel. Ils sont des simples opérations de substitution et de transposition et sont mieux adaptés pour une utilisation sur le réseau Internet filaire et en mobilité. Cependant ces systèmes ne sont pas entièrement adéquats pour résoudre tous les problèmes de sécurité.

Le processus de chiffrement et déchiffrement dépend de la même clé secrète partagée entre l'émetteur et le récepteur. L'émetteur et le récepteur doivent donc se mettre d'accord sur la clé secrète avant qu'ils puissent communiquer d'une façon sécurisée. Un canal sûr d'échange de clé est difficile à établir.

Les crypto systèmes symétriques ont un problème de gestion de clé. En effet lorsqu'un grand nombre de personnes désirent communiquer ensemble, le nombre de clés augmente de façon importante et la situation devient impraticable. Pour n participants à la communication, on aura besoin de : $n(n-1)/2$ clés secrètes enregistrés.

Il y a deux catégories de systèmes à clé privée : les chiffrements par blocs et les chiffrements de flux.

I.5.1.1 Chiffrement par bloc (Block Cipher)

C'est une des deux grandes catégories de chiffrements modernes en cryptographie symétrique. Il consiste à un découpage des données en blocs de taille généralement fixe (souvent une puissance de deux comprise entre 32 et 512 bits). Les blocs sont ensuite chiffrés les uns après les autres. Il est possible de transformer un chiffrement de bloc en un chiffrement par flot en utilisant un mode d'opération comme ECB (chaque bloc chiffré indépendamment des autres) ou CFB (on chaîne le chiffrement en effectuant un XOR entre les résultats successifs).

Un exemple de taille de bloc et de clé utilisés par les algorithmes les plus connus:

- DES : blocs de 64 bits, clé de 56 bits.
- IDEA : blocs de 64 bits, clé de 128 bits.
- AES : blocs de 128 bits, clé de 128 à 256 bits.

Pour cette catégorie, nous allons présenter deux algorithmes très connus DES et AES en mettant l'accent sur l'AES car il est devenu le standard recommandé pour le chiffrement symétrique [05].

DES (Data Encryption Sandard)

DES a été développé à la fin des années 1970s comme un standard du gouvernement US pour protéger l'information sensible. Le DES est officiellement défini dans la publication FIPS 46-3 et il est public. DES est encore supporté dans des outils de cryptographie pour les O.S des dispositifs mobiles et les cartes à puce. C'est un chiffrement qui transforme des blocs de 64 bits avec une clé secrète de 56 bits. Il a une conception basée sur le schéma de Feistel au moyen de permutations et de substitutions. L'attaque de force brute est possible sur une clé DES. Cette attaque a été réalisée par DES Cracker de EFF-Electronic Frontier Fondation en juin 1998 (Elle a trouvé une clé DES dans moins de 3 jours).

Avec la technologie actuelle (des CPU très rapides et moins chers), il est hautement possible de cracker DES. La solution a été au premier temps d'adopter le triple DES (TDES ou 3DES) : 3 applications de DES à la suite avec 2 clés différentes (112 bits) (Figure I.3).

Figure I.3:Algorithme TDES

TDES utilise une taille de blocs est 64 bits et de clé comprise entre 128 bits et 192 bits.
TDES est suffisant en sécurité mais il est trois fois plus lent que DES. Un nouvel algorithme remplace DES, c'est l'AES (Advanced Encryption Standard). [06]

AES (Advanced Encryption Standard)

En 1997, le NIST (National Institue of Standards and Technology) a lancé un appel d'offre pour un algorithme de chiffrement symétrique avec un bloc de taille 128 bits et supporte des clés de 128, 192 et 256 bits. Les critères d'évaluation de l'offre comprenaient la sécurité, la puissance de calcul, les contraintes de la mémoire des petits dispositifs (comme la carte à puce), la plateforme logicielle et matérielle et la flexibilité. Un total de 15 algorithmes ont été envoyés et 5 ont été sélectionné parmi ces 15 algorithmes [07].

L'algorithme de Rijndael a été sélectionné pour devenir l'AES en 2001. Rijndael a été conçu par Joan Daemen et Vincent Rijmen, deux chercheurs de la Belgique. Les autres algorithmes sont : Serpent, Twofish, RC6, et MARS. L'AES n'utilise pas le schéma de Feistel

mais il utilise des opérations mathématiques comme des substitutions, des permutations et des XORs. Il a plusieurs rounds identiques (de 10 à 14) et leur nombre dépend de la taille de la clé. L'AES opère au niveau octet ce qui permet une implémentation efficace au niveau matérielle et logicielle. L'AES est un standard, donc libre d'utilisation, sans restriction d'usage ni brevet. NIST spécifie actuellement AES dans le document FIPS 197, comme le nouveau standard de chiffrement symétrique. AES est approuvé pour utilisation par les organisations du gouvernement U.S pour protéger l'information sensible non classifiée.

AES a trois niveaux forts de sécurité : 128 bits, 192 bits et 256 bits. La sécurité 128 bits fournira au mois 30 ans de protection. AES ne fournit pas seulement une sécurité supérieure à TDES mais il délivre aussi une meilleure performance. Une meilleure sécurité et une meilleure performance rendent l'AES une alternative plus attractive que TDES et un bon choix pour un algorithme de chiffrement symétrique [07].

AES est également un candidat particulièrement approprié pour les dispositifs mobiles limités en ressources de calcul et de stockage. Le monde de la 3G (3ème génération de dispositifs mobiles) a adopté l'algorithme AES pour son schéma d'authentification.

Autres finalistes d'AES

Durant la compétition avec AES, Serpent et Twofish avaient une performance faible comparée à Rijndael. RC6 et MARS ont des problèmes de sécurité et d'efficacité. RC6 a été cassé à au moins de 17 rounds sur 20 pendant la compétition avec AES. MARS était plus couteux en calcul pour l'implémenter comparé aux autres finalistes d'AES [08].

I.5.1.2 Chiffrements de flux (Stream Cipher)

Les algorithmes de chiffrement de flux peuvent être définis comme étant des algorithmes de chiffrement par blocs, où le bloc a une dimension unitaire (1 bit, 1 octet, etc.) ou relativement petite. Leurs avantages principaux viennent du fait que la transformation (méthode de chiffrement) peut être changée à chaque symbole du texte clair et du fait qu'ils soient extrêmement rapides. De plus, ils sont utiles dans un environnement où les erreurs sont fréquentes car ils ont l'avantage de ne pas propager les erreurs (diffusion). Ils sont aussi utilisés lorsque l'information ne peut être traitée qu'avec de petites quantités de symboles à la fois [09]. Quelques algorithmes de cryptographie symétrique par flot:

- A5 : utilisé dans les téléphones mobiles de type GSM pour chiffrer la communication par radio entre le mobile et l'antenne-relais la plus proche.
- RC4, le plus répandu, conçu par Ronald Rivest, utilisé notamment par le protocole WEP, un algorithme récent de Eli Biham – E0 utilisé par le protocole Bluetooth.

I.5.1.3 Avantages de la cryptographie symétrique

Les avantages de la cryptographie symétrique sont:

- Système rapide du chiffrement/déchiffrement;
- Clés relativement courtes (128 ou 256 bits);
- Bonnes performances et sécurité bien étudié;

I.5.1.4 Inconvénients de la cryptographie symétrique

Les inconvénients de la cryptographie symétrique sont:

- Gestion des clés difficiles (nombreux clés) ;
- Point faible = l'échange de la clé secrète ;
- Dans un réseau de N entités susceptibles de communiquer secrètement il faut distribuer N*(N-1)/2 clés.

I.5.1.5 Considérations de sécurité pour la cryptographie symétrique

Lorsqu'on utilise la cryptographie à clé symétrique, nous devons considérer certains principes qui constituent les critères de base pour implémenter une sécurité fiable et efficace.

Choix de l'algorithme de chiffrement symétrique, la taille du bloc et la clé symétrique

Il y a plusieurs méthodes d'attaques pour trouver des vulnérabilités dans les crypto systèmes symétriques en mode bloc. Les problèmes de collision sont plus dominants dans les cryptages de bloc de 64 bits comparés à ceux de 128 bits[08]. Il est donc fondamental d'utiliser des chiffrements de bloc de 128 bits.

L'utilisation de la clé symétrique n'est pas la même pour chaque crypto système, et certaines initialisations des clés peuvent casser la sécurité offerte par l'algorithme. Par exemple, DES a un problème de clé faible et semi-faible (weak and semi-weak key). DES a 4 clés faibles et 16 clés semi-faibles. Le développeur est obligé de tester pour voir si la clé utilisée est faible ou non.

Un autre algorithme symétrique RC4 a une contrainte : la même clé ne doit jamais être utilisée pour chiffrer deux suites d'octets différentes parce que la sortie du générateur est XORé avec la suite d'octets. On trouve cette faiblesse de sécurité dans le standard WEP (Wired Equivalent Privacy) qui utilise RC4 pour chiffrer les communications sans fil sur les réseaux 802.11. Le problème avec le WEP est la taille faible de la clé. Les produits WEP implémentent une clé partagée de 64 bits, 40 bits pour la clé secrète et 24 bits pour le vecteur d'initialisation (IV-Initial Vector). Le WEP n'alloue pas suffisamment de bits pour le vecteur d'initialisation, donc la même valeur de l'IV sera réutilisée, et plusieurs paquets seront chiffrés avec la même clé[08].

Donc une règle fondamentale dans les crypto systèmes symétrique : *Chaque crypto système symétrique utilise une structure de clé différente*.

L'AES est l'algorithme le plus efficace et le plus rapide pour les plateformes logicielles et matérielles, et est le mieux approprié sur les dispositifs mobiles et PDAs. Il est donc recommandé d'utiliser l'AES pour le chiffrement/déchiffrement symétrique.

I.5.1.6 Gestion de clé

Il est fondamental d'utiliser un générateur fort de nombre pseudo aléatoire (PRNG -Pseudo Random Number Generator) pour générer des sorties d'octets aléatoires utilisées comme clés. Pour l'échange de la clé, on peut utiliser un centre de distribution de clé ou une cryptographie à clé asymétrique pour l'échange de la clé secrète. Généralement les applications utilisent un algorithme symétrique pour chiffrer les données, et un algorithme asymétrique pour chiffrer la clé secrète. Cette méthode mixte symétrique et asymétrique est mieux sécurisée et plus efficace. Si la clé est utilisée pour une communication entre deux parties sur un réseau, il est recommandé d'utiliser une clé de session parce qu'elle limite la chance à un attaquant de trouver la clé exacte. Si une clé est exigée pour chiffrer des fichiers locaux sur une machine, alors les clés générées par mot de passe (PBE-Password Based Encryption) peuvent être utilisées, mais il faut s'assurer que le mot de passe lui-même ne soit pas compromis [08]. Donc une règle fondamentale de la cryptographie symétrique : Utiliser des clés de session pour chiffrer une communication réseau et des clés PBE (basés sur des mots de passe) pour protéger les données sensibles sur la machines.

I.5.1.7 Authentifier le chiffrement symétrique

Les modes de chiffrement protègent les données de l'écoute mais ils ne fournissent aucune authentification. La fonction de déchiffrement déchiffre juste les données, elle peut déchiffrer un texte chiffré déjà modifié, en un certain plaintext qui est différent du plaintext d'origine. Dans un tel cas, il est recommandé de produire un code d'authentification du message MAC (Message Authentication Code), du texte chiffré et ajouter ce MAC au texte chiffré.

Utiliser le chiffrement en mode bloc au lieu du chiffrement par flot : Il est recommandé d'utiliser le mode de chiffrement en bloc (CBC ou CTR) avec un vecteur d'initialisation générée d'une manière aléatoire. Un vecteur d'initialisation mal choisi peut causer une faiblesse de sécurité (le cas de RC4 avec le protocole WEP).

I.5.2 Cryptographie asymétrique

La cryptographie asymétrique se base sur des problèmes mathématiques complexes (factorisation de grands nombres entiers ou équation de logarithme discrète). La cryptographie asymétrique se base sur le principe de deux clés: clé publique, clé privée (Figure I.4). La clé publique est mise à la disposition de quiconque désire chiffrer un message (cette clé peut être connu par tout le monde). Ce dernier ne pourra être déchiffré qu'avec la clé privée, qui doit être confidentielle et connue seulement par son propriétaire.

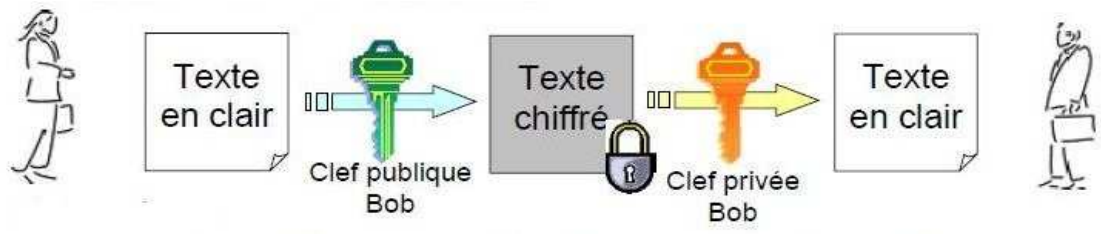


Figure I.4 : Chiffrement Asymétrique

Un chiffrement asymétrique est défini par trois algorithmes

- Algorithme de génération des clés;
- Algorithme de chiffrement;
- Algorithme de déchiffrement ;

Certains algorithmes asymétriques comme RSA offre aussi des opérations pour la génération de signature numérique et sa vérification.

L'utilisation d'une paire de clés publique/privée permet d'assurer la confidentialité, l'authentification, l'intégrité et l'échange de la clé secrète. Cependant les algorithmes asymétriques ne réalisent tous ces fonctions. La **Table I.1** donne un résumé des opérations cryptographiques pouvant être réalisés par les algorithmes asymétriques les plus connus.

Table I.1 : Les applications des crypto systèmes asymétriques

Algorithme	Chiffrement/déchiffrement	Signature numérique	Echange de clé
RSA	Oui	Oui	oui
Diffie-Hellman	Non	Non	oui
DSA	Non	Oui	non
EC-Elliptic Curves	Oui	Oui	oui

Cryptographie à clef publique basé sur des problèmes mathématiques difficiles à résoudre :

I.5.2.1 Problème de la factorisation

Le problème de la factorisation consiste à retrouver la décomposition en facteurs premiers d'un entier obtenu de manière secrète par multiplication de deux nombres premiers de taille comparable. Un tel nombre composé est classiquement appelé « module ».L'utilisation du problème de la factorisation est en général effectuée au moyen du crypto systèmes RSA [10].

I.5.2.2 Cryptage RSA

Le cryptage RSA, du nom de ses concepteurs, Ron Rivest, Adi Shamir et Leonard Adleman, est le premier algorithme de chiffrement asymétrique. Il a été découvert en 1977 au Massachusetts Institute of Technology .Le principe de ce cryptage est d'utiliser une clé publique pour crypter les données et une clé privée qui servira à les décrypter [11]. Nous allons maintenant décrire les opérations qu'on peut réaliser avec cet algorithme :

Génération des clés

Le RSA fonctionne à partir de deux nombres premiers. Ces deux nombres doivent être très grands et calculer la clé privée et la clé publique comme dans l'algorithme suivant:

Algorithme de Génération des clés

Entrées : Deux nombres premiers p et q.

Sorties : Une clé privée d et une clé publique e.

Prendre un nombre aléatoire p et q.

Si p = q alors Prendre un nombre aléatoire p et q.

Sinon N = p q.

Fin Si

Calcule $\boldsymbol{\varphi_N = (p-1)(q-1)}$.

Prendre un nombre aléatoire e dans l'intervalle [1, $\boldsymbol{\varphi_N}$].

Si $\mathbf{pgcd(e, \varphi_N) != 1}$ alors

Prendre un nombre aléatoire e dans l'intervalle [1, $\boldsymbol{\varphi_N}$].

Sinon Calcule $\mathbf{d \equiv e^{-1} mod(\varphi_N)}$.

Fin Si

Chiffrement du message

Supposons maintenant que les intervenants A et B possèdent chacun son module RSA et ses clés $N_A, e_A d_A$pour A et N_B, e_B, d_B pour B. Si A veut envoyer un message M à B, il peut procéder comme dans l'algorithme suivant:

Algorithme de Chiffrement d'un message

Entrées : Un message clair M et la clé publique $(\mathbf{N_B}, \mathbf{e_B})$.

Sorties : Un message chiffré C.

Transformer le message en un nombre entier M de l'intervalle $[\mathbf{0}, \boldsymbol{N_B - 1}]$

Calculer $\mathbf{C \equiv M^{e_B} \bmod N_B}$ et $\boldsymbol{C < N_B}$.

Envoyer le message C.

Déchiffrement du message

B a reçu un message chiffré C de la part de A. Alors B peut le déchiffrer en utilisant sa clé secrète d_B comme dans l'algorithme suivant :

Algorithme de Déchiffrement d'un message

Entrées : Un message chiffré C et la clé privée $(\mathbf{N_B}, \mathbf{d_B})$.

Sorties : Un message clair M.

Calculer $\mathbf{M = C^{d_B} \bmod N_B}$

Retourner le message M.

Signature d'un message

Supposons que A veut envoyer un message M à B. Comment B peut-il être sûr que le message reçu a bien été envoyé par A. Pour résoudre ce problème, RSA peut être utilisé aussi bien pour transmettre un message que pour le signer. Tout d'abord, A utilise la clé publique (N_B, e_B) de B pour transmettre le message chiffré C et produire une signature S du message à l'aide de sa propre clé privée (N_A, d_A) .

Algorithme de Signature d'un message

Entrées : Un message clair M et clé privée $(\mathbf{N_A}, \mathbf{d_A})$,fonction de hachage h .

Sorties : Un message chiffré C et sa signature S.

A transforme le message en un nombre entier M de l'intervalle $[\mathbf{0}, \mathbf{N_B - 1}]$.

Calculer $\boldsymbol{D = h(M)}$

Calculer $\mathbf{S \equiv D^{d_A} \bmod N_A}$

Retourner le message C et la signature S.

Vérification d'une signature

Pour vérifier la signature, nous utilisons alors l'algorithme suivant :

Algorithme de Vérification d'une signature

Entrées : Un message chiffre C, une signature S et la clé publique $(\mathbf{N_A}, \mathbf{e_A})$, et la clé privée $(\mathbf{N_B}, \mathbf{e_B})$

Sorties : Vérification d'une signature.

B déchiffre le message C : Calculer $\mathbf{M} \equiv \mathbf{C}^{d_B} \mathbf{mod}\ N_B$

B Calcule $\mathbf{D}' \equiv \mathbf{S}^{e_A} \mathbf{mod}\ N_A$ et D=h(M)

Si $\mathbf{D}' = \mathbf{D}$ Alors la signature est vérifiée.

I.5.2.3 Considérations de sécurité pour l'utilisation de RSA

- Il est fondamental d'utiliser CRT (Chinese Remainder Theorem) dans RSA pour augmenter la vitesse de déchiffrement.
- Il est aussi fondamental d'utiliser le schéma de cryptage RSAES-OAEP au lieu de RSA seul pour les opérations de chiffrement et déchiffrement.
- Il est fondamental d'utiliser une clé RSA 2048-bit comme un minimum pour l'utilisation de la cryptographie RSA dans les nouvelles applications (à partir de 2010).

I.5.2.4 Problème Logarithme discret

Le premier système du logarithme discret (DL) était le protocole d'échange de clés Diffie et Hellman en 1976. En 1984, ElGamal décrit DL chiffrement à clé publique et schémas de signature. Nous allons maintenant décrire l'algorithme qui permet :

Génération de paramètres de DL

Algorithme de génération de paramètres de DL

Entrées: paramètres t, l.

Sorties: paramètres $(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{g})$

1. Sélectionner t un nombre premier q de t-bit et un nombre premier $\boldsymbol{p}$ de l-bit tel que $\boldsymbol{q}$ divise $\boldsymbol{p} - \mathbf{1}$.
2. choisir un élément $\boldsymbol{g}$ en ordre $\boldsymbol{q}$.
3. Sélectionnez arbitraire $\boldsymbol{h} \in [\mathbf{1}, \boldsymbol{p} - \mathbf{1}]$ et Calcule $\boldsymbol{g} = \boldsymbol{h}^{(\boldsymbol{p}-\mathbf{1})/\boldsymbol{q}} \boldsymbol{mod}\ \boldsymbol{p}$.
4. Si $\boldsymbol{g} = \mathbf{1}$ alors aller à la phase 3.
5. Retourner $(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{g})$

Génération des clés DL

Algorithme de génération des clés DL

Entrées: paramètres $(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{g})$.

Sorties: clé publique $\boldsymbol{y}$, clé privée $\boldsymbol{x}$.

1. Sélectionner $\boldsymbol{x} \in \boldsymbol{R}\ [\mathbf{1}, \boldsymbol{q} - \mathbf{1}]$.
2. Calcule $\mathbf{y} = \mathbf{g}^{\mathbf{x}} \mathbf{mod}\ \mathbf{p}$.
3. Retourner $(\boldsymbol{y}, \boldsymbol{x})$.

I.5.2.5 Autres algorithmes Asymétrique

Protocole d'échange de clés DH (Diffie-Hellman) : algorithme à clé publique d'échange de clé, basé sur le problème de logarithme discret, développé par Diffie et Hellman en 1976.ce protocole permet à deux tiers de générer un secret partagé sans avoir aucune information préalable l'un sur l'autre et en échangeant uniquement leurs clés publiques.

Voici le déroulement de l'algorithme :

1. Alice choisit de manière aléatoire un grand nombre entier a, qu'elle garde secret, et calcule sa valeur publique, $A = g^a \bmod n$. Bob fait de même et génère b et $B = g^b \bmod n$;
2. Alice envoie A à Bob, Bob envoie B à Alice ;
3. Alice calcule $K_{ab} = B^a mod\ n$,Bob calcule $K_{ba} = A^b mod\ n$,$K_{ab} = K_{ba} = g^{ab} \bmod n$, c'est le secret partagé par Alice et Bob.

DSA (Digital Signature Algorithm) : Diffie-Hellman permet de crée un secret commun (et donc de chiffrer des communications) mais contrairement à RSA, il ne permet pas de signer des documents. C'est pour cette raison que Diffie-Hellman est souvent associé à DSS (Digital Signature Standard, un autre algorithme) ou DSA. DSS permet de signer les documents.

DSA est une norme promulguée par le NIST. Il est uniquement utilisé pour les signatures numériques. DSA génère des signatures plus rapides, et peut vérifier les signatures RSA.

Nous allons maintenant décrire l'algorithme qui permet :

Génération de signature de DSA

Algorithme de DSA signature generation

Entrées: DL paramètres $(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{g})$, clé prive $\boldsymbol{x}$, message $\boldsymbol{M}$.

Sorties: Signature $(\boldsymbol{r}, \boldsymbol{s})$.

1. Choisir un nombre entier $\boldsymbol{k}$ aléatoirement dans l'intervalle $]\ \mathbf{1}, \boldsymbol{q} - \mathbf{1}[$.
2. Calcule $\mathbf{T} = \mathbf{g^K mod\ p}$
3. Calcule $\mathbf{r} = \mathbf{T\ mod\ q}$. Si $\boldsymbol{r} = \mathbf{0}$ alors aller à phase 1.
4. Calcule h = *H (M)*.
5. Calcule $\mathbf{s} = \mathbf{k^{-1}}\ (\mathbf{h} + \mathbf{xr}) \mathbf{mod\ q}$. Si $\boldsymbol{s} = \mathbf{0}$ alors aller à phase 1.
6. Return $(\boldsymbol{r}, \boldsymbol{s})$.

Vérification de signature de DSA

Algorithme de DSA signature vérification

Entrées: DL paramètres $(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{g})$, clé publique $\boldsymbol{y}$, message $\boldsymbol{M}$, signature $(\boldsymbol{r}, \boldsymbol{s})$.

Sorties: Vérification d'une signature.

1. Vérifier que $(\boldsymbol{r}, \boldsymbol{s})$ soient dans l'intervalle $[\mathbf{1}, \boldsymbol{q} - \mathbf{1}]$.
2. Si n'est pas vérifier Alors return ("Signature rejetée ").
3. Calcule $\boldsymbol{h} = \boldsymbol{H}(\boldsymbol{M})$.
4. Calcule $\mathbf{w} = \mathbf{s}^{-1} \bmod \mathbf{q}$.
5. Calcule $\boldsymbol{u1} = \boldsymbol{hw} \bmod \boldsymbol{q}$ et $\boldsymbol{u2} = \boldsymbol{rw} \bmod \boldsymbol{q}$.
6. Calculer $\boldsymbol{T} = \boldsymbol{gu1} + \boldsymbol{yu2} \bmod \boldsymbol{p}$.
7. Calculer $\mathbf{r}' = \mathbf{T} \bmod \mathbf{q}$
8. Si $\mathbf{r} = \mathbf{r}'$ alors Retourner ("Signature acceptée ")

   Sinon Retourner ("Signature rejetée ").

I.5.2.6 Problème du Logarithme Discret pour les courbes elliptiques

Il est également possible de définir des problèmes de logarithme discret dans des structures plus complexes pour lesquels aucun algorithme plus efficace que les méthodes génériques de calcul de logarithme discret n'est connu. C'est en particulier aujourd'hui le cas des courbes elliptiques qui sont définies sur un corps de base pouvant être, en pratique, premier $GF(p)$ ou binaire $GF\ (2^n)$.

I.5.2.7 Avantages de la cryptographie asymétrique

Les avantages de la cryptographie asymétrique sont:

- Gestion de la secrète facilitée ;
- Pas de secrète à transmettre ;
- Nombre de clés à distribuer est réduit par rapport aux clés symétriques ;
- Très utile pour échanger les clés ;
- Permet de signer des messages facilement ;

I.5.2.8 Inconvénients de la cryptographie asymétrique

Les inconvénients de la cryptographie asymétrique sont:

La relation clé publique /clé privée impose :

- Des clés plus longues (1024 à 4096 bits) ;
- Gestion de certificats de clés publiques ;
- Lenteur de calcul;
- Pas d'authentification de la source.

I.6 Fonctions de hachage

Une fonction de hachage est aussi appelée fonction de hachage à sens unique ou "one-way hash function" en anglais est une fonction qui associe à un grand ensemble de données un ensemble beaucoup plus petit .Le résultat de cette fonction est par ailleurs aussi appelé somme de contrôle, empreinte, résumé de message, condensé ou encore empreinte cryptographique lorsque l'on utilise une fonction de hachage cryptographique. Les fonctions de HASH peuvent être utilisées pour l'authentification de message, dérivation de clé, génération de nombre pseudo aléatoire et génération de la signature numérique. Une fonction de hachage H doit avoir les propriétés suivantes :

1. H peut être appliqué à un bloc de données de toute taille.
2. H produit une sortie de longueur fixe.
3. $H(x)$ est relativement facile à calculer pour toute donnée x.
4. Pour toute empreinte (digest) d, il est informatiquement infaisable de trouver x tel que $H(x) = d$.
5. Pour tout bloc de données x, il est informatiquement infaisable de trouver $y \neq x$ avec $H(y) = H(x)$.
6. Il est informatiquement infaisable de trouver toute paire de messages (x, y) tel que $H(x) = H(y)$.

Une fonction HASH qui satisfait les 5 premiers points seulement est considérée comme faible (weak hash function), et si la sixième propriété est satisfaite, elle est alors une fonction HASH forte. Nous nous concentrons sur l'algorithme SHA (Secure Hash Algorithm) parce qu'il est le plus robuste et le plus utilisé.

I.6.1 Algorithme de hachage sécurisé SHA

- L'algorithme SHA a été conçu par NSA, standardisé par NIST et publié comme FIPS PUB 180 en 1993. Une version révisée a été délivrée en 1995 comme FIPS PUB 180-1. La version actuelle est FIPS PUB 180-2 avec une notice de changement pour SHA-224 en février 2004[12].
- La liste des algorithmes SHA est : SHA-1, SHA-256, SHA-384 et SHA-512. A cause des calculs intensifs dans SHA-384 et SHA-512, ces deux algorithmes ne sont pas recommandés pour être utilisés sur les dispositifs mobiles.
- SHA-1 est l'algorithme le plus utilisé dans la famille des algorithmes SHA. SHA-1 prend en entrée un message de taille inférieure à 2^{64} bits et produit une empreinte de 160 bits.

SHA-256 prend aussi en entrée un message de taille inférieure à 2^{64} bits mais produit une empreinte de 256 bits.

- Il y a une nouvelle attaque sur SHA-1 sans utiliser la force brute, qui trouve des collisions à l'intérieur de 269 opérations hash[13].
- A cause de l'avancement technologique et les attaques, NIST a planifié d'abandonner SHA-1 la fin 2010, et recommande des fonctions HASH plus fortes comme SHA-256.
- Les algorithmes SHA-256, SHA-384 et SHA-512 sont relativement nouveaux et sont aussi standardisés par NIST mais ils sont plus intensifs en calcul et crypto analyses.

I.6.2 Autres algorithmes de hachage

Il y a d'autres algorithmes de hachage comme MD4, MD5 et SHA-0, mais ils sont plus exposés aux attaques de collision. Il est donc recommandé d'utiliser SHA-256 comme fonction de hachage [13].

I.7 Code d'authentification de message MAC

- MAC-Message Authentification Code partage les 5 premières propriétés des fonctions de hachage à l'exception qu'il implique l'utilisation d'une clé secrète comme une entrée avec le message à authentifier. Avec MAC nous avons la fonction : $M(s,x) = d$ avec M est la fonction MAC qui génère d à partir d'un secret s et un message x. Une fonction de hachage ne peut pas être utilisée comme MAC parce qu'elle n'exige pas une clé secrète.
- Une fonction HASH peut opérer d'une manière identique à un MAC comme $H\ (s||x)$, $H(x||s)\ ou\ H(s\ ||\ x\ ||\ s)$. Il est donc fondamental d'éviter l'utilisation de fonction de hachage comme un MAC[14].
- Il y avait plusieurs approches pour incorporer une clé secrète dans un algorithme de HASH existant. L'approche la plus connue est HMAC (keyed-Hash Message Authentification Code). HMAC a été choisi pour implémenter MAC pour le protocole IPSec. Il est aussi utilisé dans d'autres protocoles comme TLS et SET (Secure Electronic Transaction).

I.7.1 HMAC

Les objectifs pour l'implémentation de HMAC :

- Utiliser sans modification des fonctions de hachage disponibles.
- Permettre de remplacer la fonction HASH intégrée dans le cas ou des fonctions HASH plus rapides, plus sécurisées sont trouvées ou exigées.
- Préserver la performance originale de la fonction HASH sans ajouter une dégradation significative.

- Utiliser et gérer des clés d'une manière simple.

HMAC est comme une boite noire qui accepte une fonction hash à intégrer comme une partie de son implémentation. L'important, si la fonction hash est craquée, elle pourra être remplacée par une autre plus sécurisée. Le HMAC sera plus sécurisée si la fonction hash intégrée est sécurisée.

HMAC fournit aussi un support pour éviter les problèmes d'extension de longueur et les collisions de message partiel[14]. La valeur de la clé secrète utilisée comme une entrée dans MAC est aussi protégée contre les attaques de recouvrement de clé.

I.7.2 Autres fonctions MAC

CBC-MAC est une méthode pour transformer un chiffrement de bloc en MAC. CBC-MAC est considérée sécurisée si le chiffrement utilisé est sécurisé. Cependant il est dangereux d'utiliser la même clé secrète pour le chiffrement CBC et l'authentification CBC-MAC[14].

UMAC est une fonction plus rapide que HMAC mais elle est limitée par une taille faible de l'empreinte (64 bits) et la complexité de ces différentes implémentations [14].

Donc HMAC est plus facile à utiliser et plus robuste comparé aux autres algorithmes MAC.

Il est recommandé d'utiliser HMAC comme fonction MAC.

I.8 Certificats à clé publique et les signatures numériques

Le format de certificat le plus largement utilisé est la version 3 du certificat X.509. Le standard ISO pour une structure de certificat numérique est basé sur le format X.509. Le format X.509 définit un Framework pour fournir des services d'authentification à travers le répertoire X.500 qui consiste à un dépôt de certificats à clé publique. Chaque certificat consiste à la clé publique d'un utilisateur et elle est signée par la clé privée d'une CA. X.509 a été initialement délivré en 1988, des recommandations de révisions ont été diffusées en 1993 pour adresser certains problèmes de sécurité avec les versions 1 et 2. La version 3 est sortie en 1995. X.509 inclut aussi des standards pour la liste de révocation de certificat (CRL). X.509 est actuellement utilisé dans les protocoles de sécurité comme SSL/TLS, IPSec, S/MIME, et SET.

I.8.1 Certificats à clé publique et le format X.509

Les certificats à clé publique X.509 sont crées par une CA de confiance et placés dans un répertoire par la CA ou par l'utilisateur. Le serveur de répertoire fournit un emplacement accessible aux utilisateurs pour obtenir les certificats. Les certificats sont codés en utilisant la syntaxe ASN.1 (Abstract Syntaxe Notation 1) de l'OSI. Les certificats utilisateur générés par la CA ont les caractéristiques suivantes [08]:

- Tout utilisateur avec l'accès à la clé publique de la CA peut récupérer la clé publique de l'utilisateur qui a été certifié.
- Aucune partie autre que la CA ne peut modifier le certificat sans qu'elle soit détectée.

Les champs du certificat X.509 [08,15] (Figure I.5):

- Version : Indique à quelle version de X.509 correspond ce certificat.
- Sérial number : Numéro de série du certificat (propre à chaque autorité de certification)
- Signature Algorithm ID : Identifiant du type de signature utilisée.
- Issuer Name : l'entité qui a signé et délivré le certificat
- Validityperiod : comprend deux dates : la date de début de validité et la date de la fin de validité.
- Subject Name : identifie l'entité associée avec la clé publique.
- Subject public key info : information sur la clé publique de ce certificat.
- Issuer Unique ID/Subject Unique ID : Extensions optionnelles introduites avec la version 2 de X.509.
- Extensions : les extensions définies pour les certificats X.509 v3 fournissant des.

Signature : la valeur du hash de tous les champs du certificat chiffrée par la clé privée de la CA [16].

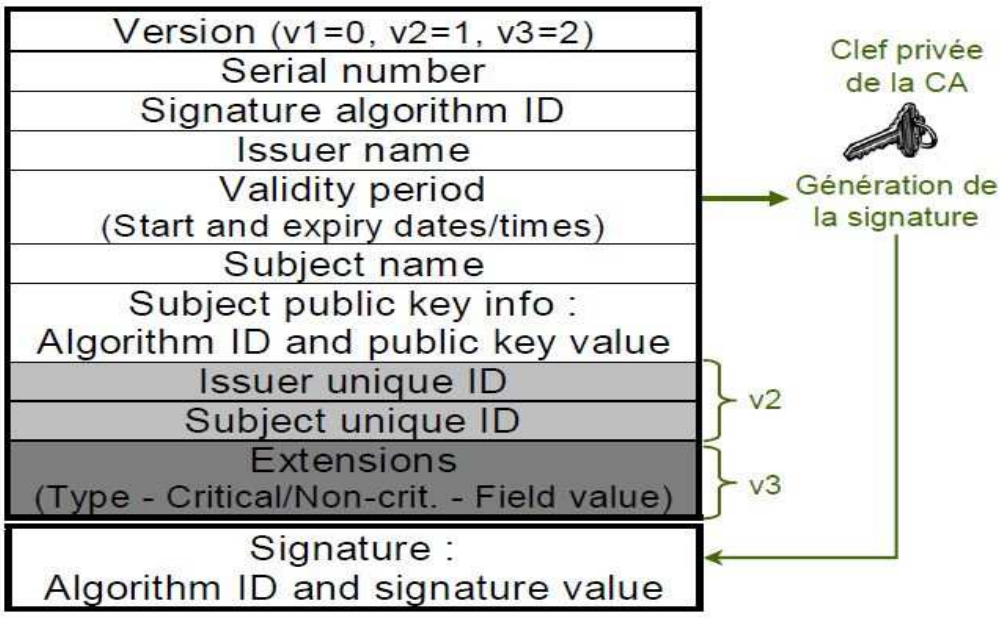


Figure I.6 : Certificat X.509

I.8.2 Signatures numériques

Les signatures numériques associées à des certificats à clé publique, fournissent l'authentification, l'intégrité et la non-répudiation. Il ya plusieurs schémas de signatures

standardisés. Le schéma PKCS#1 v2.1 supporte le cryptage RSA et il a deux schémas de signatures basés RSA : RSASSA-PSS et RSASSA-PKCS1-v1_5. Bien qu'il n'y ait pas d'attaques connues sur le schéma de signature RSASSA-PKCS1-v1_5, le standard PKCS#1 v2.1 recommande RSASSA-PSS pour les nouvelles applications à cause de sa robustesse. RSASSA-PKCS1-v1_5 est inclut dans PKCS#1 v2.1 pour compatibilité avec les applications existantes [17]. Malheureusement, le schéma de signature PSS a un brevet appliqué par l'université de la Californie, mais si PPS est inclut dans un standard IEEE et si ce standard sera adopté, une licence libre est offerte pour utiliser PSS comme schéma de signature numérique. Le schéma de signature PSS est spécifié dans IEEE P1363lorsque PKCS#1 v2.1 a été écrit. Il est recommandé d'utiliser un schéma de signature qui n'est pas protégé par un des brevets. Le schéma de signature ECDSA qui est une variante de DSA mais en utilisant les courbes elliptiques, est plus efficace que celui du RSA, Il a des tailles de clés plus petites par conséquent la signature numérique produite occupe moins d'octets que celle produite par RSA. Il est fondamental d'utiliser un algorithme de signature facile, efficace et sécurisé comme RSA ou ECDSA. Le schéma de signature ECDSA est préféré mais il n'est pas encore largement adopté.

I.9 Conclusion

L'usage ancien du chiffrement et l'usage actuel en informatique ont conduit aux contraintes suivantes :

- Réalisation simple et rapide du chiffrement et du déchiffrement.
- Eviter un encombrement important des clés.
- Un crypto systèmes dépend de paramètres (clés).

Des nouveaux services de sécurité ont été ajoutés grâce aux primitives de hachage, mac et signature numérique, la crypto moderne ajoute donc des services d'authentification, d'intégrité et de non-répudiation en plus de la confidentialité. Ces nouveaux services permettent une meilleure sécurité dans les communications et les transactions utilisant les infrastructures réseaux et Internet.

Chapitre II :

Courbes Elliptiques et Cryptographie

II.1 Introduction

L'utilisation des courbes elliptiques en cryptographie a été proposée indépendamment par N.Koblitz et V. Muller en 1985, depuis, les crypto systèmes basés sur les courbes elliptiques (Elliptic Curve Cryptosystems - ECC) ont fait l'objet d'études intensives. L'intérêt particulier qu'ils suscitent s'explique essentiellement par leur niveau de sécurité très élevé. En effet, contrairement aux autres systèmes existants, aucune attaque en temps sous exponentiel n'est connue à ce jour contre les ECC [19], sauf en des cas particuliers rares et bien identifiés.

Une autre raison qui peut expliquer l'intérêt grandissant pour ces crypto systèmes est le fait qu'ils offrent le même niveau de sécurité que d'autres crypto systèmes largement utilisés pour la signature et l'authentification (comme RSA et DSA), mais avec des clés de taille nettement inférieure. Ceci les rend très attractifs pour les applications qui nécessitent des ressources très limitées (mémoire, puissance, bande passante...) telles que cartes à puces, téléphonie mobile ou communication par satellite.

II.2 Notions algébriques

II.2.1 La structure de groupe

Un groupe G est un ensemble muni d'une loi de composition interne : $(x, y) \to x * y$ qui possède les propriétés suivantes :

1. Elle est associative, c'est-à-dire que, si x, y, z sont des éléments de G, on a : $(x * y) * z = x * (y * z)$.
2. Elle admet un élément neutre, c'est-à-dire qu'il existe un élément e $\in$ G tel que, pour tout x $\in$ G : $x * e = e * x = x$.
3. Tout élément x de G admet *un symétrique, c'est-à-dire qu'il existe un élément de G, noté* x^{-1} *, tel que :* $x^{-1} * x = x * x^{-1} = e$.

Un groupe est dit abelian (commutatif), si la loi de composition est commutative, c'est-à-dire si $x * y = y * x$ pour tout couple d'éléments de G.

II.2.2 Ordre d'un élément dans un groupe.

Si x est un élément d'un group G, l'ordre de x étant le plus petit entier positif n tel que $x^n = \underbrace{x * x * \dots * x}_{n\ fois} = e$, (on écrit $nx = e$, si la loi de G est noté additivement). Si un tel entier n n'existe pas on dit que l'ordre de x est infini.

Notons aussi que le nombre des éléments de G (si G est fini) est appelé l'ordre du groupe G.

II.2.3 Définition d'un anneau

Un anneau $\mathbb{K}$ est un ensemble muni de deux lois (+) et (*) telles que :

1. (𝕂,+) est un groupe abelian;
2. la loi (*) est associative et admet un neutre 1;
3. la multiplication est distributive par rapport à l'addition

$x * (y + z) = x * y + x * z$; $(y + z) * x = y * x + z * x$.

II.2.4 Qu'est-ce qu'un corps ?

Un corps est un anneau dans lequel tout les éléments non nul sont inversibles par rapport à la multiplication . Il revient au même dire : 𝕂 est un corps s'il est muni de deux lois (+) et (*) , satisfaisant les propriétés suivants:

1. (𝕂, +) est un groupe abelian (avec l'addition) et l'élément neutre est 0.
2. (𝕂 \ {0}, *) est un groupe dans lequel l'élément neutre est noté 1 .
3. La loi (*) est distributive par rapport à (+)

$x * (y + z) = x * y + x * z$ e $(y + z) * x = y * x + z * x$ pour tout x, y, z $\in \mathbb{K}$.

Si l'ensemble 𝕂 est fini , alors le corps est dit fini.

Notons que le groupe multiplicative d'un corps fini est toujours abelian, d'après un théorème bien connue de Wedderburn. Dans un corps 𝕂, l'ordre de 1 dans le groupe additive de 𝕂 est appelé la caractéristique de 𝕂. La caractéristique d'un corps est soit infinie ou soit un nombre premier. Les corps de caractéristique 2 sont appelés les corps binaires. Il en résulte que la caractéristique d'un corps fini 𝕂 est forcément égale à un nombre premier p. Dans ce cas, q le nombre des éléments de 𝕂 est une puissance de p, c-à-d $q = p^m$, pour certain entier positif m. La notation utilisée pour un tel corps est F_q, puisque il existe un seul corps qui a q éléments (à isomorphisme prés). En particulier $\mathbb{F}_p$ est appelé le corps premier de caractéristique p [18].

II.2.5 Définition d'une courbe elliptique

Soit 𝕂 un corps. On appelle cubique de Weierstrass E(𝕂) sur 𝕂, l'ensemble de tous les couples (x, y) dans $\mathbb{K}^2$ vérifiant une équation de la forme :

E: $Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$ avec a_1, a_2, a_3, a_4 et $a_6 \in \mathbb{K}$

Une telle équation est appelée une équation de Weierstrass. A toute équation de Weierstrass on associe les invariants suivants :

$b_2 = a_1^2 + 4a_2$

$b_4 = 2a_4 + a_1a_3$

$b_6 = a_3^2 + 4a_6$

$b_8 = a_1^2a_6 + 4a_2a_4 - a_1a_3a_4 + a_2a_3^2 - a_4^2$

$C_4 = b_2^2 - 24b_4$

$C_6 = -b_2^3 + 36b_2b_4 - 216b_6$

Bien sure ces invariants sont des éléments de $\mathbb{K}$.

Définition : Le discriminant Δ_E d'une cubique de Weierstrass E($\mathbb{K}$), est donnée par

$\Delta_E = -b_2^2b_8 - 8b_4^3 - 27\,b_6^2 + 9b_2b_4b_6$.

La cubique E($\mathbb{K}$) est dite non-singulière si son discriminant est non-nul, c.-à-d. : $\Delta_E \neq 0$.

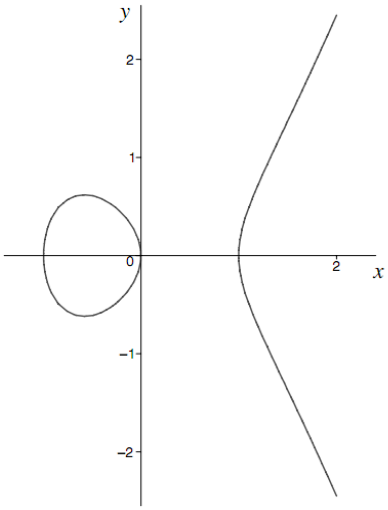
Définition. : On appelle courbe elliptique sur $\mathbb{K}$, tout couple (E, O), où E = E($\mathbb{K}$) est une cubique de Weierstrass non-singulière, et O est un point associé à la courbe, appelé le point à l'infini.

Définition : Le j-invariant j(E), d'une courbe elliptique E($\mathbb{K}$) est défini par $j(E) = \frac{C_4^3}{\Delta_E}$.

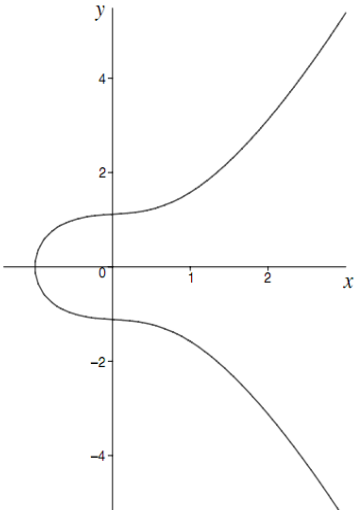
Exemples : Pour les deux cubiques suivantes (définies sur $\mathbb{Q}$, ou encore sur R),

$E_1: Y^2 = X^3 - X$ et $E_2: Y^2 = X^3 - \frac{1}{4}X + \frac{5}{4}$.

Nous avons, $\Delta_{E_1} = 64$ et $\Delta_{E_2} = -676$. Ainsi, E_1 et E_2 sont deux courbes elliptiques sur $\mathbb{Q}$ ou encore sur R. Ils peuvent être représentées dans le plan par :



$E_1: Y^2 = X^3 - X$



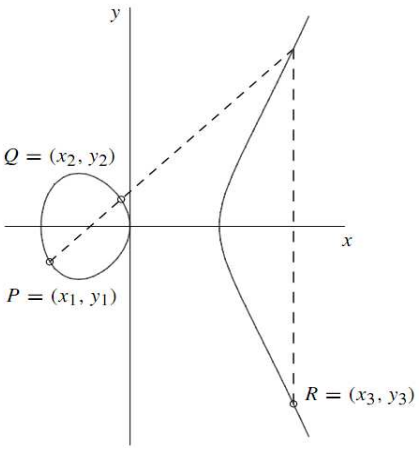
$E_2: Y^2 = X^3 - \frac{1}{4}X + \frac{5}{4}$

II.2.6 Loi de groupe

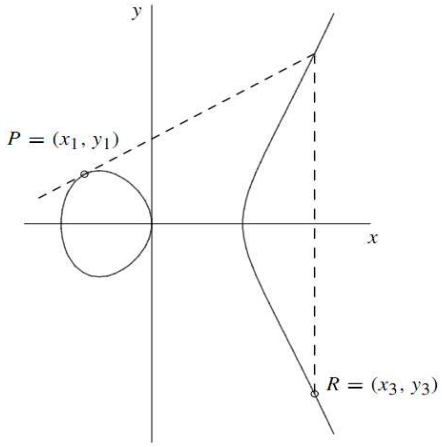
Soit E($\mathbb{K}$) une courbe elliptique, d'après un théorème profond de Bézout (effectivement, seulement un cas particulier de ce théorème), une droite qui coupe la courbe E($\mathbb{K}$) en deux points, coupe forcément E($\mathbb{K}$) en un troisième point (en comptant le point à l'infini). Ceci permet de définir une structure de groupe sur notre courbe elliptique, si P et Q sont deux points de E($\mathbb{K}$), la droite qui passe par ces deux points coupe la courbe en un troisième point X, la somme P+Q étant définie comme l'intersection de E($\mathbb{K}$) avec la droite qui passe par X et O (le point à l'infini), c-à-d la droite verticale qui passe par X.

Par construction cette loi est commutative (ceci explique la notation additive), O le point à l'infini étant le neutre pour cette loi, P et son inverse -P sont situés sur la même droite verticale. Finalement l'associativité nécessite un peu de travail.

La situation ci-dessus, peut être illustrée par :



L'Addition : P + Q =R



Le doublement : 2P = R

II.2.6.1 Formules explicites de l'addition

Bien entendue, si P est un point de la courbe P + O = O+P =P, ainsi dans la suite on suppose que notre points sont différents de O.

Soit (x_P, y_P) les coordonnées du point P de E(𝕂). Alors son opposé Q = −P a pour coordonnées : $\begin{cases} x_Q = x_P \\ y_Q = -y_p - a_1 x_P - a_3 \end{cases}$

Encore si $P(x_P, y_P)$ et $Q(x_Q, y_Q)$ sont deux points non opposés l'un de l'autre, de E(𝕂).

$R = P + Q \in E(\mathbb{K})$ de coordonnées $R(x_R, y_R)$ est donné par :

$$\begin{cases} x_R = -a_2 + \lambda^2 + a_1\lambda - x_P - x_Q \\ y_R = -(\lambda x_R + \nu) - a_1 x_R - a_3 \end{cases} \text{ où } \lambda = \begin{cases} \frac{y_Q - y_P}{x_Q - x_P} \text{ si } P \neq Q \\ \frac{3x_P^3 + 2a_2 x_P + a_4 - a_1 y_P}{2y_P + a_1 x_P + a_3} \text{ si } P = Q \end{cases} \text{ et } \nu = y_P - \lambda x_P$$

Notons que si $Q \neq -P$ et $x_P = x_Q$, alors P = Q. Par conséquent, l'addition de P et Q revient à doubler le point P.

II.2.7 Forme canonique

L'équation d'une courbe elliptique peut prendre une forme simplifiée, dite canonique. Cette forme canonique diffère légèrement suivant la caractéristique de 𝕂 :

Table II.1: Equation de weierstrass et caractéristique du corps de définition

Condition	Equation	Discriminant	j-invariant
$\mathbf{Car}(\mathbb{K}) \neq \mathbf{2,3}$	$y^2 = x^3 + a_4x + a_6$	$-16(4a_4^3 + 27a_6^2)$	$1728 \cdot 4a_4^3/(4a_4^3 + 27a_6^2)$
$\mathbf{Car}(\mathbb{K}) = \mathbf{2}$	$y^2 + a_3y = x^3 + a_4x + a_6$	a_3^4	0
	$y^2 + xy = x^3 + a_2x^2 + a_6$	a_6	$1/a_6$
$\mathbf{Car}(\mathbb{K}) = \mathbf{3}$	$y^2 = x^3 + a_4x + a_6$	$-a_4^3$	0
	$y^2 = x^3 + a_2x^2 + a_6$	$-a_2^3\, a_6$	$-a_2^3\, /a_6$

II.2.8 Courbes elliptiques définies sur un corps fini

II.2.8.1 Cardinalité

Définition : Le nombre de points du groupe $E(\mathbb{F}_q)$, appelé cardinalité de la courbe elliptique et noté $Card(E(\mathbb{F}_q))$, est le nombre de solutions de l'équation de Weierstrass. Vu comme un polynôme de $\mathbb{F}_q[Y]$, l'équation de Weierstrass est du second degré. Il en résulte que pour chaque valeur de x, il y a au plus deux valeurs de y telles que le couple (x, y) vérifie cette équation. Puisque $x \in \mathbb{F}_q$, il y a q valeurs possibles pour x, et donc en comptant O, au plus $2q + 1$ couples sont solutions de l'équation de Weierstrass. En moyenne on a une chance sur deux pour que, x étant fixé, l'équation en Y admet des solutions dans $\mathbb{F}_q$, plus précisément on a :

Théorème de Hasse : Soit $E(\mathbb{F}_q)$, où $q = p^n \in \mathbb{N}$, une courbe elliptique. On a alors [19]:

$$|card\left(E(\mathbb{F}_q)\right) - (q+1)| \leq 2\sqrt{q}.$$

Ce théorème ne fournit qu'un encadrement du nombre de points de la courbe. Or en cryptographie, il est essentiel de connaître le nombre précis de points de la courbe elliptique manipulée. Des algorithmes ont donc été construits dans le but de connaître le nombre exact de points d'une courbe elliptique.

II.3 Courbes elliptiques et cryptographie

II.3.1 Introduction

Les courbes elliptiques destinées aux applications cryptographiques doivent être définies sur un corps premier $\mathbb{F}_P$ ou sur un corps binaire fini$\mathbb{F}_{2^n}$.

Les crypto systèmes utilisant les courbes elliptiques définies sur $\mathbb{F}_{2^n}$ doivent se plier à certaines exigences pour garantir une meilleure sécurité. De telles courbes sont de moins en

moins utilisées car le corps $\mathbb{F}_{2^n}$ est considéré comme trop structuré. Cependant les calculs sur de tels crypto systèmes ont l'avantage d'être plus faciles à implémenter.

Les crypto systèmes utilisant les courbes elliptiques définies sur les corps finis de la forme $\mathbb{F}_P$ peuvent être compromis par l'attaque (eg : MOV). La taille des clés utilisées doit alors respecter les critères de sécurité appliqués au problème du logarithme discret dans un corps fini, on perd alors l'intérêt d'utiliser les courbes elliptiques [19].

II.3.2 Courbes elliptiques sur $\mathbb{K} = \mathbb{F}_p$

Dans le cadre des courbes elliptiques définies sur le corps $\mathbb{F}_p$, avec une caractéristique différente de 2 et 3, Les variables et coefficients prennent des valeurs dans l'ensemble $[0, p-1]$ pour un certain nombre premier$p$, et où toutes les opérations sont calculées modulo p. L'équation devient $y^2 = x^3 + ax + b$.

II.3.2.1 Formules explicites des opérations

Pour tous points $P, Q \in \mathbb{F}_p\ (a, b)$:

- P+O = P ;

Si $P(x_1, y_1)$ et $Q(x_2, y_2)$ alors $-P = (x_1, -y_1)$, Si $Q = -P$ alors $P + Q = O_E$

sinon $P + Q = R(x_3, y_3)$ où $\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases}$ Avec $\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \text{ si } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} \text{ si } P = Q \end{cases}$

II.3.3 Courbes elliptiques sur $\mathbb{K} = \mathbf{F_{2^n}}$

Dans le cadre des courbes elliptiques définies sur $\mathbb{F}_{2^n}$, corps de caractéristique 2, il existe deux formes simplifiées de l'équation de Weierstrass :

$$y^2 + xy = x^3 + a_2x^2 + a_6 \bmod(2^n)$$
$$y^2 + a_3y = x^3 + a_4x + a_6 \bmod(2^n)$$

II.3.3.1 Formules explicites des opérations

Opposé : Soit $P(x_P, y_P)$ un point d'une courbe elliptique $E(\mathbb{F}_{2^n})$. Son opposé Q a pour coordonnées : $\begin{cases} x_Q = x_P \\ y_Q = x_P + y_P \end{cases}$

Addition : Soient $E(F_{2^n})$ une courbe elliptique convenable sur $\mathbb{F}_{2^n}$, deux points de $E(\mathbb{F}_{2^n})$, $P(x_P, y_P)$, $Q(x_Q, y_Q)$ non opposés. Alors si $R = (x_R, y_R) = P + Q$:

$$\begin{cases} x_R = \lambda^2 + \lambda + x_P + x_Q + a_2 \\ y_R = (\lambda + 1)x_R + \lambda x_P + y_P \end{cases} \quad \text{Avec} \quad \lambda = \begin{cases} \frac{y_P + y_Q}{x_P + x_Q} \text{ si } P \neq Q \\ \frac{x_P^2 + y_P}{x_P} \text{ si } P = Q \end{cases}$$

II.4 Problème du logarithme discret (ECDLP)

Un problème sur lequel des protocoles à clé publique, sont basés. Soit G un groupe (noté additivement) cyclique fini d'ordre N engendré par un élément P. Soit Q un élément de G. Comme G est engendré par P, il existe un entier unique n compris entre 1 et N tel que Q = nP. Cet entier n est appelé le logarithme discret de Q en base P et est noté $n = \log_P(Q)$. Usuellement ce problème est plutôt présenté pour un groupe noté multiplicativement ce qui donne : Soit G un groupe (noté multiplicativement) cyclique fini d'ordre N engendré par un élément g. Soit h un élément de G. Comme G est engendré par g, il existe un entier unique n compris entre 1 et N tel que $h = g^n$. Cet entier n est appelé le logarithme discret de h en base g et est noté $n = log_g(h)$.

L'avantage de cette définition est qu'on comprend mieux d'où vient le nom du logarithme discret puisque si h et g sont des réels, on a bien $n = \frac{log(h)}{log(g)}$ c'est à dire le logarithme de h en base g.

II.5 Paramètres de domaines

Les paramètres de domaine D tels que $D = (q, a, b, P, n, h)$ sont composés de :

1. L'ordre du corps q.
2. Deux coefficients $a, b \in \mathbb{F}_q$ qui déniassent l'équation de la courbe elliptique E sur $\mathbb{F}_q$ (i.e.,$y^2 = x^3 + ax + b$ dans le cas d'un corps premier ou d'extension optimal, et $y^2 + xy = x^3 + ax^2 +$ b dans le cas d'un corps binaire).
3. Deux éléments du corps $x_p \, et \, y_p$ dans $\mathbb{F}_q$ qui définissent un point fini $P = (x_p, y_p) \in E(\mathbb{F}_q)$ en coordonnées affines. P a un ordre premier et est appelé le point de base.
4. L'ordre n de P.
5. Le cofacteur $h = \frac{card\,(E(\mathbb{F}))}{n}$.

II.6 Problèmes liés à l'utilisation des courbes elliptiques en cryptographie

Nous allons le voir par la suite, l'implémentation des opérations sur une courbe elliptique varie suivant le choix de la représentation utilisée pour définir cette courbe. L'efficacité des algorithmes de calcul de l'ordre d'un point dépend de cette implémentation. Ce choix est donc primordial. Naturellement la sélection de la courbe elliptique est elle aussi décisive pour assurer un niveau de sécurité suffisant [19].

II.6.1 Généralités sur l'implémentation des opérations

Dans cette partie présenter les algorithmes de calculs sur les points d'une courbe elliptique. Quel que soit le corps de définition d'une courbe elliptique, les formules explicites des

opérations sur les points sont similaires. Pour plus de clarté nous étudierons donc le cas d'une courbe définie sur F_{2^n}, les notations associées étant plus simples. Soit $E(\mathbb{F}_{2^n})$ une courbe elliptique représentée en coordonnées affines (ou non-homogènes) et donnée par l'équation: $y^2 + xy = x^3 + a_2x^2 + a_6$.

Lorsque le modèle de Weierstrass est utilisé, l'addition définie sur une courbe elliptique est décomposée en deux cas suivant que l'on additionne deux points distincts ou qu'on double un point. Deux algorithmes sont donc employés pour l'addition.

Voici deux algorithmes effectuant chacun une de ces opérations sur la courbe elliptique $E(\mathbb{F}_{2^n})$,le premier additionnant deux points distincts, le second doublant un point :

Addition de deux points distinct

Entrées : $\mathbf{P}(\mathbf{x_P}, \mathbf{y_P}), \mathbf{Q}(\mathbf{x_Q}, \mathbf{y_Q}) \in \mathbf{E}(\mathbb{F}_{\mathbf{2^n}})$

Sorties : R = P + Q =$(\mathbf{x_R}, \mathbf{y_R}) \in \mathbf{E}(\mathbb{F}_{\mathbf{2^n}})$

Début

$\mathbf{X} \leftarrow \mathbf{x_P} + \mathbf{x_Q}$;

$\boldsymbol{\lambda} \leftarrow \frac{\mathbf{y_P}+\mathbf{y_Q}}{\mathbf{X}}$;

$\mathbf{x_R} \leftarrow \boldsymbol{\lambda}^{\mathbf{2}} + \boldsymbol{\lambda} + \mathbf{X} + \mathbf{a_2}$;

$\mathbf{y_R} \leftarrow (\boldsymbol{\lambda} + \mathbf{1}) \cdot \mathbf{x_R} + \boldsymbol{\lambda} \cdot \mathbf{x_P} + \mathbf{y_P}$;

Retourner R = $(\mathbf{x_R}, \mathbf{y_R})$;

Fin

Double d'un point

Entrées : $\mathbf{P}(\mathbf{x_P}, \mathbf{y_P}) \in \mathbf{E}(\mathbb{F}_{\mathbf{2^n}})$

Sorties : R = 2P =$(\mathbf{x_R}, \mathbf{y_R}) \in \mathbf{E}(\mathbb{F}_{\mathbf{2^n}})$

Début

$\boldsymbol{\lambda} \leftarrow \mathbf{x_P} + \frac{\mathbf{y_P}}{\mathbf{x_P}}$;

$\mathbf{x_R} \leftarrow \boldsymbol{\lambda}^{\mathbf{2}} + \boldsymbol{\lambda} + \mathbf{a_2}$;

$\mathbf{y_R} \leftarrow \mathbf{x_P^2} + \boldsymbol{\lambda} \cdot \mathbf{x_R} + \mathbf{x_R}$;

Retourner R = $(\mathbf{x_R}, \mathbf{y_R})$;

Fin

Le calcul de l'opposé d'un point d'une courbe elliptique peut être réalisé par l'algorithme suivant :

Opposé d'un point

Entrées : $\mathbf{P}(\mathbf{x_P}, \mathbf{y_P}) \in \mathbf{E}(\mathbb{F}_{2^n})$

Sorties : R = -P =$(\mathbf{x_R}, \mathbf{y_R}) \in \mathbf{E}(\mathbb{F}_{2^n})$

Début

$\mathbf{x_R} \leftarrow \mathbf{x_P}$;

$\mathbf{y_R} \leftarrow \mathbf{x_P} + \mathbf{y_P}$;

Retourner R = $(\mathbf{x_R}, \mathbf{y_R})$;

Fin

L'algorithme suivant, qu'utilise ceux décrits précédemment, permet de traiter l'addition dans un cadre plus général :

Addition de deux points

Entrées : $\mathbf{P}(\mathbf{x_P}, \mathbf{y_P}), \mathbf{Q}(\mathbf{x_Q}, \mathbf{y_Q}) \in \mathbf{E}(\mathbb{F}_{2^n})$

Sorties : R = P + Q =$(\mathbf{x_R}, \mathbf{y_R}) \in \mathbf{E}(\mathbb{F}_{2^n})$

Début

Si (P = O) alors

Retourner $\mathbf{Q} = (\mathbf{x_Q}, \mathbf{y_Q})$ FinSi ;

Si (Q = O) alors

Retourner $\mathbf{P} = (\mathbf{x_P}, \mathbf{y_P})$ FinSi ;

Si (P = Opposé d'un point (Q)) alors

Retourner O FinSi ;

Si (Q = P) alors

Retourner Double d'un point(P) ;

Sinon Addition de deux points distinct FinSi ;

Retourner R = $(\mathbf{x_R}, \mathbf{y_R})$;

Fin

Un autre calcul, très important dans le cadre de la cryptographie, est celui qui permet la multiplication d'un point par un entier. En effet, cette opération est indispensable pour appliquer le problème du logarithme discret. Voici un algorithme qui effectue ce calcul :

Multiple d'un point

Entrées : $\mathbf{k} = (\mathbf{k_t}, \cdots\cdots\cdots, \mathbf{k_1}, \mathbf{k_0}) \in \mathbb{N}, \mathbf{P}(\mathbf{x_P}, \mathbf{y_P}) \in \mathbf{E}(\mathbb{F}_{2^n})$

Sorties : Q = kP =$(\mathbf{x_R}, \mathbf{y_R}) \in \mathbf{E}(\mathbb{F}_{2^n})$

Début

$\mathbf{Q} \leftarrow \mathbf{O}$;

Pour i de k à 0 faire $\mathbf{Q \leftarrow 2Q}$;
Si $\mathbf{k_i = 1}$ **alors** $\mathbf{Q \leftarrow Q + P}$;
Fin Si
Fin Pour
Retourner Q ;
Fin

Comme nous pouvons le constater, le calcul de l'addition de deux points ainsi que le doublement d'un point consiste à effectuer un inverse, deux multiplications et une élévation au carré. Cependant, calculer l'inverse d'un élément de F_{2^n} est obtenu à l'aide d'un algorithme dont la complexité est élevée.

II.6.2 Calcul de l'ordre d'un point et d'une courbe elliptique

II.6.2.1 Calcul de l'ordre d'un point P

Déterminer les coordonnées d'un point P d'une courbe elliptique E(𝕂) revient à fixer aléatoirement un élément x de 𝕂 et vérifier que l'équation de Weierstrass définissant cette courbe, alors ramenée à une équation du second degré, admet une solution $y \in \mathbb{K}$. De par la structure de corps de 𝕂, les formules de résolution des racines d'un trinôme sont valables, On peut donc facilement déterminer $y \in \mathbb{K}$. connaissant les coordonnées d'un point P et la factorisation en produit de facteurs premiers de l'ordre de E(𝕂), il est possible de déterminer l'ordre de P en temps polynomial. Voici un algorithme qui nous permet un tel de calcul :

Order d'un point

Entrées : $\mathbf{P(x_P, y_P) \in \# E(F_{2^n})}$,$\mathbf{E = p_1^{e_1} \cdots\cdots p_r^{e_r}}$
Sorties : $m \in \mathbb{N}$
Début
$\mathbf{m \leftarrow \# E}$;
Pour i de 1 à r faire
$\mathbf{m \leftarrow m/p_i^{e_i}}$; $\mathbf{Q \leftarrow mP}$;
Si $\mathbf{P \neq O}$ **alors**
$\mathbf{Q \leftarrow P_iQ}$;
$\mathbf{m \leftarrow mP_i}$;
Fin Si
Fin Pour
Retourner m ;
Fin

II.6.2.2 Calcul de l'ordre d'une courbe E(𝕂)

Le théorème de Hasse sur le nombre de points d'une courbe elliptique fournit l'approximation: $q + 1 - 2\sqrt{q} \leq \text{card}(E(Fq)) \leq q + 1 + 2\sqrt{q}$.

II.7 Application des courbes elliptiques à la cryptographie

II.7.1 ECIES (Elliptic Curve Integrated Encryption Scheme)

C'est un système de chiffrement, variante de l'algorithme ElGamal à clé publique. Il a été standardisé par ANSI X9.63 et ISO/IEC 15946-3, il est aussi défini dans le standard IEEE P1363. Dans ECIES, un secret partagé de type Diffie-Hellman est utilisé pour dériver deux clés symétriques k1 et k2. La clé k1 est utilisée pour chiffrer le texte clair en utilisant un algorithme de chiffrement symétrique, k2 est utilisé pour authentifier le texte chiffré résultant. pour chiffrer un message clair m, on l'encode comme un point dans une courbe elliptique. Si Alice veut envoyer un message secret à Bob en utilisant l'algorithme ECIES, l'échange se passe comme suit :

Chiffrement d'un message

Entrées: paramètres $\mathbf{D} = (\mathbf{q}, \mathbf{a}, \mathbf{b}, \boldsymbol{P}, \mathbf{n}, \mathbf{h})$, clé public Q, message M.

Sorties : Un message chiffré $(\boldsymbol{R}, \boldsymbol{C}, \boldsymbol{t})$.

1. Choisir un nombre entier secret $\mathbf{k} \in \mathbf{R}\ [\mathbf{1}, \mathbf{n} - \mathbf{1}]$.
2. Calcule $\boldsymbol{R} = \boldsymbol{kP}$ **et** $\boldsymbol{Z} = \boldsymbol{hkQ}$, Si $\boldsymbol{Z} = \infty$ Alors aller à 1.
3. Calcule $(\boldsymbol{k1}, \boldsymbol{k2}) = \boldsymbol{KDF}(\boldsymbol{xZ}, \boldsymbol{R})$, où $\boldsymbol{xZ}$ est coordonnée de Z.
4. Calcule $\boldsymbol{C} = \boldsymbol{ENC}\ (\boldsymbol{k1}, \boldsymbol{m})$ et $\boldsymbol{t} = \boldsymbol{MAC}\ (\boldsymbol{k2}, \boldsymbol{C})$.
5. Retourner $(\boldsymbol{R}, \boldsymbol{C}, \boldsymbol{t})$.

MAC : un code authentification de message.

KDF(key derivation function) : fonction de génération de clé .

ENC : un algorithme de chiffrement comme(AES).

Bob déchiffre le message de la manière suivante :

Déchiffrement d'un message

Entrées: paramètres $\mathbf{D} = (\mathbf{q}, \mathbf{a}, \mathbf{b}, \boldsymbol{P}, \mathbf{n}, \mathbf{h})$, clé public Q, clé privé d, message chiffré $(\boldsymbol{R}, \boldsymbol{C}, \boldsymbol{t})$.

Sorties : Un message clair M.

1. Choisir un nombre entier secret $\mathbf{k} \in \mathbf{R}\ [\mathbf{1}, \mathbf{n} - \mathbf{1}]$.
2. Calcule Z = h d R , Si $\boldsymbol{Z} = \infty$ Alors retourner ("texte chiffré rejeté").
3. Calcule $(\boldsymbol{k1}, \boldsymbol{k2}) = \boldsymbol{KDF}(\boldsymbol{xZ}, \boldsymbol{R})$, où $\boldsymbol{xZ}$ est coordonnée de *Z*.
4. Calcule t'= MAC (k2,C)
5. Si $\boldsymbol{t'} \neq \boldsymbol{t}$ Alors retourner ("texte chiffré rejeté")
6. Retourner $\boldsymbol{M} = \boldsymbol{DEC}\ (\boldsymbol{k1}, \boldsymbol{C})$.

DEC : un algorithme de déchiffrement.

II.7.2 Echange de clés de Diffie-Hellman

ECDH (Elliptic Curve Diffie Hellman) est un protocole d'échange de clés secrètes. Deux entités Alice et Bob souhaite disposer d'une clé secrète commune. Ils se mettent d'accord sur le choix d'une courbe elliptique $E(\mathbb{K})$ où $\mathbb{K}$ est un corps fini, et sur le choix d'un point P de cette courbe. Ces choix sont connus de tous. L'échange d'une clé par ECC entre deux entités se déroule comme suit [20] :

Echange de clés de Diffie-Hellman ECDH

Entrées : **courbe** $\mathbf{E}(\mathbb{K})$, $\mathbf{P} \in \mathbf{E}(\mathbb{K})$

Sorties : clé secrète commune K

1. Alice choisit un secret a, Bob choisit un secret b Ces entiers constitueront leurs clés privées ;
2. Alice calcule sa clé publique $\mathbf{A} = \mathbf{aP}$, Bob calcule sa clé publique $\mathbf{B} = \mathbf{bP}$;
3. Alice envoie alors à Bob le point *A* et celui-ci lui envoie le point *B* ;
4. Alice effectue alors le calcul $\mathbf{aB} = \mathbf{abP}$ et Bob effectue alors le calcul $\mathbf{bA} = \mathbf{abP}$;
5. Retourner K=abP ;

Si quelqu'un, que nous appellerons Oscar, espionne leurs communications et intercepte les points *A* et *B*, le problème du logarithme discret garantit qu'il ne sera pas en mesure de déterminer les entiers *a* et *b*. Il ne pourra donc pas reconstituer la clé *abP* commune à Alice et Bob. Oscar dispose cependant d'une manière d'espionner ces conversations s'il est en mesure de substituer un nouveau message à celui d'Alice puis à celui de Bob :

La courbe $E(\mathbb{K})$ et le point P étant connus de tous, il peut choisir un entier *c* et calculer le point $C = cP$.

Oscar intercepte le message d'Alice, récupère le point *A* et le remplace par *C*. De même il intercepte le message de Bob, récupère le point *B* et le remplace par *C*. Il calcule alors les points $QA = cA = caP$ et $QB = cB = cbP$. De leurs côtés Alice et Bob ont reçu tous les deux le point $C = cP$ et ont alors calculé respectivement les points $aC = acP = QA$ et $bC = bcP = QB$.

Lorsqu'Alice envoie un message à Bob, elle le chiffre alors avec la clé *QA*.

Oscar intercepte ce message, le déchiffre car il est en possession de la clé *QA*, puis le rechiffre à l'aide de la clé *QB*. Il envoie le message chiffré, modifié ou non, à Bob qui le déchiffre grâce à sa clé *QB*. Oscar doit alors intercepter toutes les conversations entre Alice et Bob pour ne pas que ceux-ci s'aperçoivent de sa présence. En effet ne disposant pas de clé commune, Alice et Bob ne sont plus en mesure de déchiffrer ces messages sans l'intervention d'Oscar.

La faiblesse de ce protocole réside donc dans le fait qu'il ne permet pas d'authentifier les auteurs des messages émis.

II.7.3 ECDSA (Elliptic Curve Digital Signature Algorithm)

Le schéma de signature électronique ECDSA (Elliptic Curve Digital Signature Algorithm) dont le fonctionnement est similaire à la signature DSA. Le mécanisme DSA (Digital Signature Algorithm) repose sur le problème du logarithme discret sur les corps finis, il peut donc être appliqué aux courbes elliptiques. L'intérêt de cette pratique provient du fait que, à un niveau de sécurité équivalent, l'utilisation des courbes elliptiques permet des calculs plus rapides et réclame moins de mémoire par rapport à l'utilisation d'un corps fini. En effet, pour un niveau de sécurité équivalent, cet algorithme travaille avec des clés de plus petite taille, par exemple 160 bits au lieu de 1024 pour le DSA classique, tout en conservant les tailles des signatures [20].

Le schéma de signature électronique EC-DSA se résume à quatre algorithmes :

1. Un algorithme de génération des paramètres de domaine D.
2. Un algorithme de génération des clés à partir de D et généré une paire de clés (Q, d).
3. Algorithme de génération de signature à partir d'entrée D, clé privé d, et message m.
4. Algorithme de vérification de signature à partir d'entrée D, clé public Q, et message m, la signature.

La mise en place du schéma EC-DSA nécessite une paire de clés, l'une publique, l'autre privée. La clé publique est accessible à tous et permet à chacun de vérifier l'intégrité du message et l'authenticité de l'entité qui l'a envoyé.

Alice souhaite envoyer un message *m* signé par le protocole EC-DSA à Bob. Pour cela elle va choisir une paire de clé (clé publique, clé privée) en procédant comme suit :

Préparation des clés

Entrées : paramètres de domaine $\mathbf{D} = (\mathbf{q}, \mathbf{a}, \mathbf{b}, \boldsymbol{P}, \mathbf{n}, \mathbf{h})$

Sorties : la clé publique Q et sa clé privée d.

1. Elle Choisit un entier d entre 1 et n – 1 ;
2. Elle Calcule $\boldsymbol{Q} = \boldsymbol{dP}$;
3. Sa clé publique sera Q et sa clé privée $\boldsymbol{d} = \boldsymbol{log_P(Q)}$;

On remarque que c'est le problème du logarithme discret de base *P* qui garantit la difficulté de déterminer la clé privée connaissant la clé publique *Q*.

Alice dispose maintenant de la paire de clés dont elle a besoin. Pour signer son message elle procède ainsi :

Génération d'une signature ECDSA

Entrées: paramètres de domaine $\mathbf{D} = (\mathbf{q}, \mathbf{a}, , \boldsymbol{P}, \mathbf{n}, \mathbf{h})$, clé prive d,un message M.

Sorties : Signature $(\boldsymbol{r}, \boldsymbol{s})$.

1. Choisit de manière aléatoire un nombre $\boldsymbol{k} \in [\mathbf{1}, \boldsymbol{n} - \mathbf{1}]$.
2. Calcule $\boldsymbol{kP} = (\boldsymbol{x1}, \boldsymbol{y1})$ et convertir $\boldsymbol{x1}$ en un entier $\overline{\mathbf{x1}}$.
3. Calcule $\boldsymbol{r} = \overline{\mathbf{x1}} \ \mathbf{mod\ n}$. Si $\boldsymbol{r} = \mathbf{0}$ alors elle recommence.
4. Calcule $\boldsymbol{e} = \boldsymbol{H}(\boldsymbol{m})$.
5. Calcule $\mathbf{s} = \mathbf{k}^{-1}(\mathbf{e} + \mathbf{dr}) \ \mathbf{mod\ n}$. Si $\boldsymbol{s} = \mathbf{0}$ alors elle recommence.
6. Retourner $(\boldsymbol{r}, \boldsymbol{s})$.

Bob reçoit le message m signé par le couple (u,v), il doit :

Vérification d'une signature ECDSA

Entrées: paramètres de domaine $\mathbf{D} = (\mathbf{q}, \mathbf{a}, , \boldsymbol{P}, \mathbf{n}, \mathbf{h})$, clé public Q, message m, signature *(r, s)*.

Sorties :Acceptation ou rejet de la signature.

1. Si $\boldsymbol{r} \in [\mathbf{1}, \boldsymbol{n} - \mathbf{1}]$ **ou** $\boldsymbol{s} \in [\mathbf{1}, \boldsymbol{n} - \mathbf{1}]$ alors retourner (‘’ rejet de la signature’’) fin si ;
2. Calcule $\boldsymbol{e} = \boldsymbol{H}(\boldsymbol{m})$.
3. Calcule $\boldsymbol{w} = \boldsymbol{S^{-1}mod\ n}$.
4. Calcule $\boldsymbol{u_1} = \mathbf{ew\ mod\ n}$ **et** $\boldsymbol{u_2} = \mathbf{rw\ mod\ n}$.
5. Calcule $\boldsymbol{X} = \boldsymbol{u_1}\mathbf{P} + \boldsymbol{u_2}\mathbf{Q}$.
6. Convertir la coordonnée $\boldsymbol{x1}$ de $\boldsymbol{X}$ en un entier $\overline{\mathbf{x1}}$; calculer $\mathbf{v} = \overline{\mathbf{x1}} \ \mathbf{mod\ n}$.
7. Si $\boldsymbol{v} = \boldsymbol{r}$ alors retourner (“Acceptation de la signature”);

   Sinon retourner(“Rejeter de la signature”).

II.8 Avantages et Inconvénient

	Asymétrique(RSA)	Asymétrique(ECC)
Avantages	- Crypto système largement répandu - Nombreuses études au sujet de sa sécurité	- Taille de clé inférieure pour une sécurité égale - La taille des clés croit moins vite que le RSA si on souhaite une meilleure sécurité - Utilisation pour systèmes embarqués - Utilisation mémoire moindre - plus rapids - plus puissant
Inconvénients	- Opérations de dé/chiffrement très inégales en termes de temps de calcul - Cryptanalyse par algorithme sous exponentiel	- Complexe - Peu de développement sur des systèmes à grande échelle (mais tend à changer) - Travaux d'optimisation essentiellement destinés aux systèmes mobiles

II.9 Comparaison des tailles des clés

C'est là le gros avantage des courbes elliptiques par rapport à RSA puisque pour un niveau de sécurité équivalent, RSA nécessite l'utilisation de clé de plus de 1024 bits. De plus, le rapport de taille devient de plus en plus important à mesure que le niveau de sécurité augmente. Ainsi une clé ECC de 256 bits est aussi robuste qu'une clé RSA de 3072 bits. Pis encore, avoir le niveau de sécurité d'une clé ECC de 512 bits requiert l'utilisation de clés RSA de 15360 bits. Dans le tableau 1.4 nous donnons différentes tailles de clé ayant des niveaux de sécurité équivalente sur différents algorithmes (Table II.2) [21].

Table II.2 : Comparaison de tailles de clé à niveau de sécurité équivalent

Sécurité (en bits)	RSA ou Diffie-Hellman	courbes elliptiques
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

II.10 Conclusion

Les crypto systèmes basés sur les courbes elliptiques se posent en alternative efficace face à l'incontournable RSA. En effet, ils exploitent un problème mathématique différent, qui est réputé pour sa solidité égale à RSA pour des clés de longueur bien inférieure. Cela les rend parfaitement adaptés aux utilisations embarquées, comme les cartes à puce par exemple, où la mémoire et la puissance des processeurs ne sont pas suffisants pour réaliser en un temps convenable les calculs exigés par RSA. De nos jours la cryptographie est en perpétuelle évolution afin de pouvoir répondre aux besoins de sécurisation des données qui ne cessent d'augmenter. En effet, les crypto systèmes se doivent d'être performants face à des attaques de plus en plus nombreuses. C'est pourquoi nous ne pouvons pas prédire combien de temps les crypto systèmes basés sur les courbes elliptiques seront les plus efficaces au niveau sécurité.

Chapitre III :

L'API de Sécurité Java et L'architecture JCA/JCE

III.1 Introduction

La plateforme Java a été conçue sur une base de sécurité. Le langage Java lui-même est de type sécurisé. La plateforme a évoluée et a élargi son intervalle de déploiement, l'architecture de la sécurité java a aussi évolué pour supporter un ensemble de services cryptographiques.

Actuellement, l'architecture comprend un ensemble très large d'APIs, d'outils, d'implémentations d'algorithmes, et de protocoles de sécurité. Java fournit au développeur un Framework de sécurité pour écrire des applications, et aussi à l'utilisateur ou l'administrateur un ensemble d'outils pour la gestion sécurisée des applications.

Les APIs de sécurité Java couvrent plusieurs secteurs. Les interfaces de cryptographie et de PKI (Public Key Infrastructure). Ces interfaces vérifiant l'authentification et le contrôle d'accès, permettent aux applications de se protéger contre les accès non autorisés aux ressources [22]. Pour les services de cryptographie, nous présentons l'architecture de cryptographie JCA et son extension JCE. Nous montrons les principes de conception de JCA/JCE, la notion de CSP (Cryptographic Service Provider). Nous présentons les concepts introduits dans l'API de sécurité comme les classes de moteur (engine classes) et l'interface de provider de service SPI (Service Provider Interface) pour l'implémentation des providers de service. Dans la classe Provider, nous montrons comment installer et configurer un provider et comment utiliser ses implémentations ? Nous prenons comme cas pratique le provider JCE de Bouncy Castle, nous montrons son installation et sa configuration dans la plateforme J2SE.

III.2 Sécurité du langage Java

Le langage Java est conçu pour être de type sécurisé et facile à utiliser. Le langage définit plusieurs modes d'accès qui peuvent être affectés aux classes, aux méthodes et aux variables Java. Il définit quatre niveaux d'accès différents: private, protected, public, et package. Le mode d'accès le plus utilisé est public pour une utilisation ouverte et sans restriction. Le mode le plus restrictif est private qui n'est pas accessible en dehors de la classe. Le mode protected permet l'accès à toute classe ou sous classe à l'intérieur du même package. Le niveau d'accès package permet seulement l'accès aux classes à l'intérieur du même package [22].

Le compilateur Java produit un programme Java au format byte-code qui s'exécute sur une machine virtuelle Java indépendamment de la machine et de la plateforme OS. Un vérificateur du byte-code est utilisé automatiquement pour s'assurer que le programme exécuté dans

l'environnement d'exécution Java (JRE) est en byte-code. Une fois le byte-code est vérifié, l'environnement d'exécution Java le prépare pour l'exécution [22].

III.3 Cryptographie dans Java

III.3.1 API

Les APIs permettent plusieurs implémentations interopérables d'algorithmes et d'autres services de sécurité. Les services sont implémentés dans des providers. Ces providers sont ajoutés dans la plateforme via une interface standard qui facilite aux applications d'utiliser ces services de sécurité sans connaitre les détails de leurs implémentations.

III.3.2 Concepts cryptographiques implémentés par l'API de sécurité Java

La spécification de l'API de sécurité Java utilise des termes techniques pour désigner des concepts cryptographiques. On donne la signification exacte de ces termes utilisés [23]:

- Chiffrement et déchiffrement (Encryption and Decryption): Le chiffrement est le processus qui prend des données en entrée (plaintetxt) et une clé (key) pour produire des données chiffrées (ciphertext). Le déchiffrement est le processus inverse qui prend le texte chiffré et une clé pour reproduire les données originales en texte clair.
- Chiffrement basé mot de passé (PBE-Password-Based Encryption) : Le chiffrement basé mot de passe (PBE) permet de créer une clé de chiffrement à partir d'un mot de passe. Pour éviter les attaques de mot de passe (ou attaque de dictionnaire).
- Agrément de clé (Key Agreement) ou établissement de clé : C'est un protocole qui permet d'établir des clés cryptographiques entre deux ou plusieurs parties.
- Code d'authentification de message (MAC-Message Authentification Code) : Un code d'authentification de message (MAC) fournit un moyen pour vérifier l'intégrité de l'information transmise ou stockée dans un endroit incertain, basé sur une clé secrète.

  Le mécanisme MAC qui est basé sur une fonction de hachage et un secret, est désigné sous le nom *HMAC*. Un *HMAC* peut être utilisé avec toute fonction de hachage, e.g. MD5 ou SHA1, en combinaison avec une clé secrète partagée.

III.3.3 Services cryptographiques fournis

L'architecture de la cryptographie Java est un Framework pour l'accès et le développement de fonctionnalités cryptographiques. Elle comprend des APIs pour un grand nombre de services cryptographiques, comprenant[1]:

- Algorithmes d'empreinte de message
- Algorithmes de signature numérique
- Chiffrement en mode bloc *(block cipher)*

- Chiffrement symétrique en mode flux de données (streamcipher)
- Chiffrement basé mot de passe (PBE)
- Cryptographie de courbe elliptique (ECC)
- Algorithmes d'agrément de clé
- Générateurs de clés secrètes et de paire de clés
- Codes d'authentification de message (MACs)
- Générateurs de nombres pseudo aléatoires (PRNG)

A cause des contrôles d'exportation imposés par le gouvernement U.S sur les logiciels de cryptographie, les APIs Java cryptographiques sont organisés dans deux packages différents :

- Le package java.security contient des classes non concernées par les contrôles d'exportation (comme digital signature et message digest).
- Le package javax.crypto contient des classes concernées par les contrôles d'exportation (comme Cipher et KeyAgreement).

III.4 Architecture JCA/JCE

III.4.1 Présentation des architectures JCA et JCE

L'API de sécurité fait partie du noyau du Java. Elle est conçue pour permettre aux développeurs d'incorporer des fonctionnalités de sécurité dans leurs programmes.

III.4.1.1 JCA (Java Cryptography Architecture)

L'API de sécurité dans JDK 1.1 a introduit l'architecture de la cryptographie Java-JCA (Java Cryptography Architecture), qui est un Framework pour l'accès et le développement de fonctionnalités cryptographiques pour la plateforme Java. JCA comprenait des APIs pour signatures numériques et empreintes de messages. C'est une architecture basée provider permettant d'implémenter un ou plusieurs providers de services cryptographiques interopérables, connus comme providers de services cryptographiques- CSPs (Cryptographic Service Providers). L'API JCA s'appuie sur le package *java.security* [24].

III.4.1.2 JCE (Java Cryptography Extension)

JCE est une API qui propose de standardiser l'utilisation de la cryptographie en restant indépendant des algorithmes utilisés. Elle prend en compte le chiffrement/déchiffrement de données, la génération de clés et l'utilisation de la technique MAC (Message Authentification Code) pour garantir l'intégrité d'un message.

JCE a été intégrée au JDK 1.4. Pour pouvoir utiliser cette API, il faut obligatoirement utiliser une implémentation développée par un fournisseur (provider). Avec le JDK 1.4, Sun

fournit une implémentation de référence nommée SunJCE. Les classes et interfaces de l'API JCE sont regroupées dans le package *javax.crypto*.[24]

JCA/JCE sont architecturés pour fournir une couche d'abstraction pour les développeurs d'application, les objets fournis par l'implémentation (les algorithmes) sont crées en utilisant des classes de production (factory classes).

Cette architecture est basée provider, ce qui signifie que JCA/JCE fournissent une implémentation d'un ensemble de classes et d'interfaces qui ne sont pas directement visibles au développeur. JCA/JCE ont des mécanismes simples pour permettre aux développeurs d'ajouter et de choisir des providers particuliers.

La Figure III.1 montre comment les différents composants fonctionnent ensemble. Le code de l'application fait appel aux classes appropriées des APIs JCA/JCE, ces dernières appellent les classes dans un provider qui fournit des implémentations pour les classes SPI. Les classes utilisent le code interne dans le provider pour fournir la fonctionnalité demandée [24].

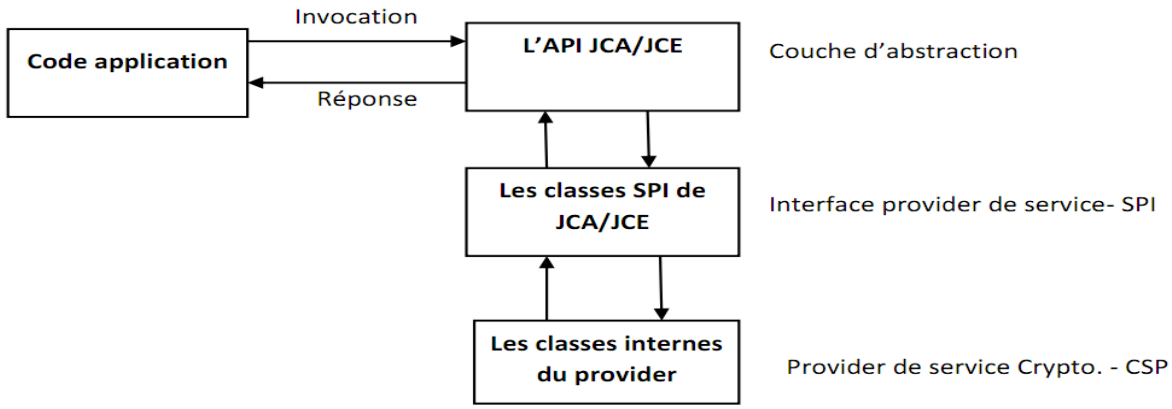


Figure III.1 : L'API JCA/JCE et les providers de service

Sun a introduit l'utilisation des fichiers de restrictions « policy files » qui permettent l'utilisation des algorithmes et les tailles de clés avec restriction, il a aussi introduit l'idée de Providers signés. Les changements sont signés par l'agence nationale de sécurité NSA (National Security Agency) et le département de commerce U.S.

III.4.2 Principes de conception des providers dans JCA/JCE

JCA/JCE a deux principes de conception de providers [24]:

- Leur indépendance de l'implémentation et leur interopérabilité : On peut utiliser les services cryptographiques sans connaitre les détails de l'implémentation.
- Leur indépendance des algorithmes et leur extensibilité : des nouveaux providers de services et/ou des algorithmes peuvent être ajoutés sans affecter les providers existants.

Ces deux principes fournissent une architecture modulaire ce qui permet de réaliser un chiffrement par une implémentation et le déchiffrement par une autre implémentation. L'indépendance de l'implémentation et l'indépendance de l'algorithme sont complémentaires, on peut utiliser des services cryptographiques comme les signatures numériques et les empreintes de message sans connaitre les détails de l'implémentation ou les algorithmes qui forment la base de ces concepts. L'indépendance de l'algorithme est réalisée par la définition des types de service cryptographiques « Moteurs » et la définition des classes qui fournissent la fonctionnalité de ces moteurs cryptographiques. Ces classes sont appelées engine classes. L'indépendance de l'implémentation est réalisée par l'utilisation de l'architecture basée « provider ». Le CSP (Cryptographic Service Provider) se réfère à un package ou un ensemble de packages qui implémentent un ou plusieurs services cryptographiques tels que les algorithmes de signatures numériques et les empreintes de messages. Un programme peut simplement demander un type particulier d'objet (comme un objet de signature) implémentant un service particulier (comme l'algorithme de signature DSA) et obtenir une implémentation d'un des providers installés. Les providers doivent afficher leur mise à jour d'une manière transparente à l'application, par exemple quand des versions plus rapides ou plus sécurisées sont disponibles.

III.5 Concepts introduits dans L'API de sécurité

Les concepts principaux introduits dans l'API de sécurité sont :

III.5.1 Classes de moteur et les algorithmes

Une classe de moteur (engine class) définit un service cryptographique d'une manière abstraite sans aucune implémentation concrète. Un service cryptographique est généralement associé avec un algorithme particulier qui permet de [23]:

- fournir des opérations cryptographiques (comme les signatures numériques ou les empreintes de messages),
- générer ou fournir les clés ou les paramètres pour les opérations cryptographiques, ou générer des objets de données (certificats et keystore) qui enveloppent les clés cryptographiques utilisées dans les opérations de cryptographie.

Par exemple Signature et KeyFactory sont deux classes de moteurs. La classe Signature fournit l'accès à la fonctionnalité de l'algorithme de signature numérique. La classe KeyFactory DSA fournit une clé DSA privée ou publique d'un objet de signature DSA, dans un format utilisable respectivement par les méthodes initSign ou initVerify.

JCA/JCE contient les classes du package de sécurité Java liées à la cryptographie, y compris les classes de moteur. Les utilisateurs de l'API utilisent des instances de ces classes de moteur pour réaliser les opérations correspondantes.

L'application interface fournie par la classe de moteur est implémentée en termes de SPI. Pour chaque classe de moteur, il y a une classe SPI abstraite correspondante pour définir les méthodes que le CSP doit implémenter. Pour fournir l'implémentation d'un type particulier de service, pour un algorithme spécifique, un provider doit créer une sous classe de la classe SPI correspondante et fournir l'implémentation de toutes les méthodes de la classe abstraite.

Une instance d'une classe de moteur (l'objet API) encapsule une instance de la classe SPI correspondante (l'objet SPI). L'instance de la classe de moteur (et sa classe SPI correspondante) est crée en appelant la méthode de production getInstance de la classe de moteur. Le nom de chaque classe SPI est le même que celui de sa classe de moteur suivi par Spi. Par exemple, la classe SPI correspondante à la classe de moteur Signature est SignatureSpi. La sous classe SignatureSpi permet de spécifier le type de l'algorithme de signature tel que SHA1withDSA, SHA1with RSA, ou MD5withRSA.

III.5.2 Implémentations et Providers

Le package du provider JCA par défaut de Java (appelé SUN) inclut les services cryptographiques d'authentification comme les empreintes de message (SHA1) et les signatures numériques (DSA). Le provider Java JCE de Sun (appelé SunJCE) fournit un framework pour le chiffrement, la génération et l'agrément de clé, et les algorithmes MAC- (Message Authentication Code). Le support de chiffrement comprend les chiffrements symétriques et asymétriques, en mode bloc et flux de données (block, and stream cipher) [23].

NB : Support de l'algorithme ECC (Elliptic Curve Cryptogaphy) Avec la version J2SE 5 et supérieure, des interfaces et des classes ECC ont été ajoutées pour faciliter l'ajout de providers supportant la cryptographie ECC [23].

III.6 La classe Provider

La classe Provider est l'interface des packages des CSPs. Elle a des méthodes pour accéder aux informations de provider (nom, numéro de version et autres informations). Pour fournir des implémentations de services cryptographiques, un groupe de développement écrit le code de l'implémentation et crée une sous classe de la classe Provider. Le constructeur de la sous classe Provider positionne les valeurs des différentes propriétés, l'API de sécurité de Java 2 SDK utilise ces valeurs pour rechercher les services implémentés par ce provider. En d'autres termes, la sous classe spécifie les noms des classes implémentant les services [23].

Il y a plusieurs types de services qui peuvent être implémentés par les packages du provider. Les implémentations peuvent avoir des caractéristiques différentes. Certaines peuvent être basées programmes, alors que d'autres peuvent être basées matériels. Certaines peuvent être indépendantes de la plateforme, d'autres peuvent être spécifiques à des plateformes. Le code source de certains providers peut être disponible pour révision et évaluation, alors que d'autres providers n'affichent pas leur code source. L'architecture de la cryptographie java laisse aux utilisateurs et aux développeurs de décider de leur besoins.

III.6.1 Comment utiliser les implémentations d'un provider ?

Pour chaque classe de moteur dans l'API, une implémentation particulière est instanciée par l'appel à une méthode de production (factory method) sur la classe de moteur. Une méthode de production est une méthode statique qui retourne une instance d'une classe. Par exemple, le mécanisme de base pour obtenir un objet Signature est comme suit :

```
Signature   signature = Signature.getInstance("SHA1withRSA", "BC");
```

La méthode getInstance recherche une implémentation qui satisfait l'algorithme spécifié et les paramètres du provider (son nom, ici BC). Si aucun provider n'est spécifié, getInstance recherche par ordre de préférence, un provider parmi les providers installés, qui a une implémentation de l'algorithme spécifié. Dans la machine virtuelle Java (JVM), les providers sont installés dans un ordre de préférence. Un programme peut avoir la liste de tous les providers installés (en utilisant la méthode getProviders() de la classe Security), et choisir un provider de la liste.

III.6.2 Installation et configuration de providers

Pour plus d'informations sur l'implémentation d'un provider, voir le guide « How To Implement a Provider for the Java Cryptography Architecture » à la page Web :http://download.oracle.com/javase/6/docs/technotes/guides/security/crypto/HowToImplAProvider.html.

Il ya deux étapes pour installer un provider : installer les classes de package du provider et configurer le provider [23] :

- **Installer les classes du provider**

Il ya deux façons d'installer les classes du provider :

- Placer le fichier zip ou JAR contenant les classes dans le chemin de la variable d'environnement CLASSPATH.

- Déployer le fichier JAR des classes du provider comme une extension de Java (Package optionnel).

Pour plus d'information sur comment déployer une extension ? voir « How is an extension deployed?» à la page Web :

http://download.oracle.com/javase/6/docs/technotes/guides/extensions/spec.html.

- **Configurer le Provider**

L'étape suivante est d'ajouter le provider à la liste des providers enregistrés. On doit configurer et enregistrer le provider pour permettre aux clients d'utiliser son service. Il y a deux façons pour enregistrer un provider :

- **Enregistrement statique :** on ajoute le nom du provider à la liste des providers existants dans le fichier java.security du sous répertoire lib/security du JRE. Pour chaque provider disponible il y a une ligne correspondante de la forme :

```
security.provider.n= masterClassName
```

Cette ligne permet de déclarer un provider et spécifier son ordre de préférence n. L'ordre de préférence est celui dans lequel les providers sont recherchés pour les algorithmes demandés (lorsqu'aucun provider n'est spécifié). L'ordre est basé 1 : 1 est le plus préféré, suivi de 2 et ainsi de suite.

masterClassName doit spécifier la classe maitre du provider. Cette classe est toujours une sous classe de la classe Provider. Le constructeur de sous classe positionne les valeurs des propriétés qui sont nécessaires pour l'API de cryptographie Java pour chercher les algorithmes implémentés par le provider.

- **Enregistrement dynamique :** peut être réalisé par l'application client qui appelle des méthodes de la classe Security telle que : Security.addProvider, ou insertProviderAt. Ce type d'enregistrement n'est pas persistant et peut être seulement réalisé par des programmes sûrs et de confiance.

Chaque instance de classe Provider a un nom, un numéro de version, et une chaine décrivant le provider et ses services. La classe Provider a des méthodes pour avoir ces informations, on peut interroger l'instance Provider en appelant les méthodes suivantes : getName(), getVersion(), et getInfo().

III.6.3 Gestion des Providers

La classe Security gère les providers installés et les propriétés de sécurité. Elle contient uniquement des méthodes statiques et elle n'est jamais instanciée. Les méthodes pour ajouter

ou supprimer des providers, et pour positionner les propriétés de sécurité, peuvent seulement être exécutées par un programme de confiance « a trusted program » [23]. On utilise les méthodes de la classe Security pour interroger les providers installés, et pour aussi installer ou enlever des providers au moment de l'exécution.

III.7 Provider Bouncy Castle

III.7.1 Définition Bouncy Castle

Bouncy Castle est un package (ou bibliothèque de cryptographie) libre et open source. Il ressemble à la librairie C openssl qui est conforme aux différents standards en vigueur. Bouncy Castle n'est pas installé de base sur les plateformes java. Il est nécessaire d'avoir des notions en cryptographie. Car même si des explications seront apportées sur les méthodes, elles ne seront peut être pas suffisantes pour une compréhension totale. Le package BC peut être téléchargé du site http://www.bouncycastle.org/ dans un seul module comprenant le provider JCE.

III.7.2 Présentation du package crypto de Bouncy Castle (BC)

Le package crypto de Bouncy Castle est une implémentation d'algorithmes cryptographiques pour la plateforme Java. Les algorithmes du package BC sont conformes au Framework JCE. Lorsque l'accès aux librairies JCE n'est pas possible dans certains cas (comme le cas des applets Java ou les MIDlets Java exécutés sur les petits dispositifs avec ressources limitées), le package BC contient une API légère (lightweight) appropriée a l'environnement mobile comme la plateforme J2ME/MIDP. L'API BC existe aussi en version C# et fournit les même fonctionnalités que celle du Java.

Bouncy Castle est Australien d'origine donc il n'est pas concerné par les restrictions américaines sur l'exportation des logiciels cryptographiques.

III.7.3 Spécifications du package BC

Le package BC inclut [25]:

- Une implémentation de l'API JCE
- Une API cryptographique Légère supportant les algorithmes cryptographiques suivants: BlockCipher,BufferedBlockCipher,AsymmetricBlockCipher,BufferedAsymmetricBlockCipher,StreamCipher,BufferedStreamCipher,KeyAgreement,IESCipher,Digest,Mac et PBE.
- Un Framework JCE compatible au provider BC

III.7.4 Provider JCE Bouncy Castle

Le provider Bouncy Castle est un provider compatible JCE. Il supporte les types d'algorithmes cryptographiques suivants [25] :

- Chiffrement symétrique en mode bloc (*avec ou sans schémas de remplissage-padding*) et en mode flux de données, plusieurs algorithmes symétriques sont supportés tels que : AES, Blowfish, Camellia, DES, DESede, IDEA, RC2, RC4, RC5, RC6, Twofish,...
- Chiffrement asymétrique (RSA, ElGamal) avec les schémas de codage suivants :
  - ✓ OAEP - Optimal Asymmetric Encryption Padding,
  - ✓ PCKS1 - PKCS v1.5 Padding,
  - ✓ ISO9796-1 - ISO9796-1 edition 1 Padding,
- Empreinte de message tels que:GOST3411,MD2,MD4, D5, RipeMD128/160/256/320, SHA1/224/256/384/512, Tiger, Whirlpool
- Code d'authentification de message-MAC comme: HMac-MD2/MD4/MD5,HMac-RipeMD128/160,HMacSHA1 /224/256/384/512, HMac-Tiger,…
- Chiffrement basé mot de passe PBE
- Agrément de clé (key agreement) avec les algorithmes DH et ECDH
- Une implémentation de l'algorithme ECIES comme décrit dans IEEE P 1363a.
- Des signatures numériques telles que : GOST3411withGOST3410, MD2withRSA, MD5withRSA,SHA1withRSA,SHA1withDSA,SHA1withECDSA,SHA224withECDSA,SHA256withECDSA,SHA384withECDSA,SHA512withECDSA,SHA224withRSA, SHA256withRSA,SHA384withRSA, SHA512withRSA, …

BC supporte aussi les services suivants [25]:

- Le standard des certificats X.509 : Le provider Bouncy Castle peut lire les certificats X.509 (v2 ou v3) avec la classe *java.security.cert.CertificateFactory*. Ces certificats peuvent être fournis dans le format d'encodage PEM ou comme des les fichiers binaires de type DER. La classe *CertificateFactory* lit aussi les listes de révocations de certificats X.509 (CRLs) de formats PEM ou DER. En plus des classes dans le package *org.bouncycastle.asn1.x509* pour la génération de certificat, une autre classe JCE est fournie dans le package *org.bouncycastle.x509*, et elle supporte les certificats RSA, DSA et ECDSA.
- Magasin de clés ou Keystore: Le package de Bouncy Castle a trois implémentations de keystore pour le stockage permanent des certificats et des clés privées.
- Support de classes pour les courbes elliptiques EC : Le package BC a aussi des classes supplémentaires pour supporter les courbes elliptiques (clés et paramètres EC).Nous trouvons ces classes dans les packages suivants: *org.bouncycastle.jce.spec, org.bouncycastle.jce.interfaces, etorg.bouncycastle.jce.*

- Pour plus d'informations voir le site :
  http://www. Bouncycastle.org/Java/Specifications.html.

III.7.5 Choix de Bouncy Castle

Nous avons utilisé le package BC avec l'architecture JCA/JCE de Java pour plusieurs raisons : BC est open source, riche en algorithmes crypto, n'est pas aussi concerné par les règles de restriction imposées sur les packages US, et son développement est continu, actuellement on est à la version 1.48, ces algorithmes sont corrigés et vérifiés par des milliers de développeurs à travers le monde. BC implémente aussi la cryptographie de courbes elliptique. BC a également une API légère qui peut être utilisé dans les environnements ou l'accès aux librairies JCE n'est pas possible (comme le cas des applets Java téléchargés et exécutés sur le poste client ou les MIDlets Java de la plateforme J2ME destinés aux petits dispositifs mobiles contraints en ressources)

Cependant BC a aussi des inconvénients comme le manque de documentation, l'absence de manuels d'utilisation de ces algorithmes, certains algorithmes comme la génération de clés et les paramètres d'initialisation sont très complexes à utiliser. Il est donc difficile à utiliser par des développeurs débutants en java et en techniques de crypto. Malgré ces inconvénient, BC demeure le meilleur package Open Source pour les développeurs d'applications crypto dans la plateforme Java et nous l'avons choisi pour implémenter nos benchmarks de comparaison d'opération crypto basés sur courbes elliptiques.

III.7.6 Installer le provider JCE de Bouncy Castle dans la plateforme Java:

III.7.6.1 Installer le fichier JAR contenant le provider

Le fichier JAR peut être téléchargé de la page web:
http://www.bouncycastle.org/latest_releases.html, Si on utilise JDK 1.6 de la plateforme J2SE, on cherche le fichier JAR du provider BC à la version 1.45, les noms des fichiers JARs à télécharger sont **bcprov-jdk16-145.jar et bcprov-ext-jdk16-145.jar**. Télécharger ces fichiers et les copier dans le répertoire *jre/lib/ext* de l'installation Java [24] .

III.7.6.2 Activer le provider en l'ajoutant dans le fichier java.security

Le répertoire *jre/lib/security* contient les fichiers des contrôles des restrictions *java.policy ,javaws.policy* et *java.security*. Ce dernier spécifie les providers qui sont actifs pour JCA/JCE et leur ordre de préférence (priorité). On observe les lignes suivantes du fichier :

```
# List of providers and their preference orders (see above):
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=com.sun.security.sasl.Provider
security.provider.7=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.8=sun.security.smartcardio.SunPCSC
security.provider.9=sun.security.mscapi.SunMSCAPI
```

Ajouter la ligne suivante à la fin de la liste :

```
security.provider.10=org.bouncycastle.jce.provider.BouncyCastleProvider
```

III.7.6.3 Vérifier l'installation de provider Bouncy Castle

Le programme **Provider_test.java**, pourra indiquer si le provider BC est installé en affichant le message suivant :

```
Provider BC est installé.
```

III.7.6.4 Afficher les providers installés

La classe *java.security*. Security a une méthode statique getProviders() qui permet d'avoir tous les providers installés. Pour afficher le nom du provider et son n° de version, la classe java.security.Provider fournit les méthodes getName() et getVersion(). On peut utiliser un petit programme java pour afficher les providers installés avec leurs noms et leurs versions :

```
Name: SunRsaSign      Version: 1.5
Name: SunJSSE         Version: 1.6
Name: SunJCE          Version: 1.6
Name: SunJGSS         Version: 1.0
Name: SunSASL         Version: 1.5
Name: XMLDSig         Version: 1.0
Name: SunPCSC         Version: 1.6
Name: SunMSCAPI       Version: 1.6
Name: BC              Version: 1.45
```

III.7.6.5 Ordre de préférence des providers

Le fichier de configuration *java.security* a des numéros de préférence (priorités) associés aux providers installés. Lorsqu'on instancie un objet JCA/JCE par la méthode de production *getInstance()*,on peut soit laisser Java runtime choisir pour nous le provider, soit spécifier le provider qu'on veut utiliser comme suit :

```
Cipher.getInstance("Blowfish/ECB/NoPadding", "BC");
```

Si on ne spécifie pas le provider à la création de l'objet, on écrira :

```
Cipher.getInstance("Blowfish/ECB/NoPadding");
```

Le provider avec le numéro de préférence le plus faible a la priorité la plus élevée. On utilise le programme suivant test_prec.java, on aura la sortie suivante:

```
SunJCE version 1.6
BC version 1.45
```

Le provider SunJCE est choisi avant BC, sa priorité dans le fichier java.security est supérieure à celle de BC, il est le premier provider dans la liste qui peut satisfaire ce service crypto.

III.7.6.6 Capacités d'un provider

Les providers rendent leurs capacités disponibles au JCA/JCE en utilisant une table de propriété publique, ainsi il est préférable d'écrire un programme qui affiche les possibilités fournies par un provider donné. Un petit programme Java affiche les noms de l'algorithme de base et les classes de production pouvant être utilisés avec le provider BC. Le programme utilise la méthode Security.getProvider() pour retrouver un provider en fournissant son nom. Il peut afficher la classe de production et le nom de l'algorithme fourni dans cette classe. Ce ci permet d'avoir une idée claire sur les services cryptographiques fournis par ce provider (**Table III.1**).

Table III.1: Services cryptographiques java fournis par le provider BC

Service Crypto.	Algorithme/Type	Description
KeyGenerator	BLOWFISH,DES,DESEDE, AES	Générer des clés secrètes utilisées par ces algorithmes
KeyPairGenerator	RSA,DH,ECDSA,ECDH,ECDHC, DSA, ECIES	Générer une paire de clés publique/privée
MessageDigest	SHA1, MD5,…	Calculer l'empreinte d'un message
Mac	HMAC/SHA1, HMACSHA256,	Calculer le code d'authentification d'un message

	HMACMD5	
Signature	SHA1WithRSA,SHA1withECDSA,SHA256WITHDSA,SHA256withECDSA,SHA1withDSA, MD4withRSA	Créer et vérifier la signature numérique d'un message
KeyStore	BouncyCastle, PKCS12, UBER	Stocker des clés et des certificats
CertificateFactory	X.509	Créer des ertificats
Cipher	RSA/OAEP,AES,EWITHSHA-1AND192BITAES-CBC-BC	Chiffrer et déchiffrer des messages
KeyAgreement	DH, ECDH, ECDHC, ECMQV	Permettre à deux parties d'agréer une clé secrète sans l'échanger sur un canal non sécurisé

Pour produire un objet JCA/JCE du provider BC, on utilise la méthode getInstance(), l'objet peut être Cipher, MessageDigest, Mac, SecretKeyFactory et ainsi de suite. Dans le cas d'un objet MessageDigest SHA1,l'appel à la méthode MessageDigest.getInstance()est comme suit :

```
MessageDigest digest = MessageDigest.getInstance("SHA-1", "BC");
Signature signature = Signature.getInstance("SHA1withRSA", "BC");…
```

Pour avoir les noms des algorithmes implémentés dans les classes Cipher, keyAgreement, MAC, MessageDigest et Signature, nous pouvons utiliser le programme liste_algorithmes.java.

III.8 Conclusion

Dans ce chapitre, nous avons montré l'évolution de l'API de sécurité java, les concepts de sécurité introduits dans java , l'architecture de cryptographie Java JCA/JCE et ses principes de conception. Nous avons aussi montré comment installer et configurer un provider pour pouvoir utiliser ses services cryptographiques. Bouncy Castle est un provider de services crypto open source, et riche en algorithmes cryptographiques. Nous avons montré comment installer et configurer ce provider dans l'environnement d'exécution Java ? comment vérifier son installation ? quels sont ses algorithmes fournis ? comment fonctionne la méthode de production getInstance() ? et comment l'ordre de préférence des providers est utilisé ? Les développeurs peuvent utiliser les providers implémentés dans la plateforme Java ou ajouter un provider tierce partie Open Source comme Bouncy Castle. Durant les dernières années, Java est devenue une technologie avec une croissance rapide. Les possibilités cryptographiques ont été ajoutées à la plateforme Java SE avec l'architecture JCA/JCE. Ces possibilités sont

étendues en partie à d'autres plateformes comme J2ME. Dans java SE, à partir de la version 5, il est possible d'utiliser des implémentations ECC adaptées au framework JCA/JCE, mais il est aussi possible d'utiliser des APIs légères comme celle de Bouncy Castle.

L'architecture JCA/JCE a les avantages suivants :

- Les fonctionnalités cryptographiques sont disponibles d'une façon gratuite via une API standard.
- Les clients et les utilisateurs finaux auront un code de base commun, ainsi la cryptographie n'a pas besoin d'être incluse dans des packages commerciaux.

Mais elle a aussi des inconvénients :

- La plateforme Java devient plus complexe à adopter, ses services cryptographiques sont difficiles à utiliser (plusieurs packages, plusieurs méthodes et des chemins compliqués).
- Certains algorithmes cryptographiques ne sont pas fournis en standard.

Chapitre IV :

Implémentation Et Réalisation des Benchmarks

IV.1 Introduction

Au cours de ce chapitre, nous allons présenter des Benchmarks utilisés pour la comparaison de deux familles de courbes GF_p et GF_{2^m} , définis par les Standards de cryptographie à clé publique comme NIST et SECG.

Nous avons implémenté des benchmarks qui testent la performance des opérations cryptographiques sur notre PC. Nous avons utilisé, le provider de service crypto de Bouncy Castle sous la plateforme Java. Les tests ont été réalisés par répétion des opérations cryptographiques sur un texte court contenant des informations sensibles. Pour le nombre d'itérations, nous avons utilisé un nombre de l'ordre de 1e6 (1 000 000) itérations pour les opérations cryptographiques rapides comme le chiffrement et le déchiffrement ECIES. Pour les opérations cryptographiques plus lentes comme la signature numérique et sa vérification ECDSA, et le calcul du secret ECDH, nous avons utilisé un nombre de l'ordre de 1e3 (1 000) itérations.

Les opérations de génération de paire de clés du crypto système asymétrique consomment du temps, donc pour les paires de clés ECIES, ECDSA et ECDH, nous avons utilisé 1000 itérations.

Nous allons présenter en premier l'environnement de notre travail : la plateforme Java et l'éditeur de développement intégré utilisé.

IV.2 JDK

JDK est un logiciel (ensemble des outils officiels) édité par SUN pour le développement nécessaires à la création et à l'exécution de programmes Java. Ils contiennent notamment les compilateurs javac, l'interpréteur java, l'outil d'archivage jar, et plein d'autre choses. Ils sont en générale fourni avec le code source de l'API et pas mal de fichiers d'aides. Dans ce travail nous avons utilisée JDK 1.6[26].

IV.3 Editeur

Il existe plusieurs EDI sur le marché comme par exemple, Jbuilder, Jcreator , Netbeans ou encore Eclipse. Nous avons choisi Netbeans pour notre développement dans le langage Java.

Netbeans est un environnement de développement intégré (EDI). C'est une application qui propose dans le même système de fenêtrage, des outils qui facilite le travail au développeur. Par exemple, cet environnement propose :

- Un éditeur de texte avec coloration syntaxique des éléments clés du langage.
- De la même façon, il propose l'auto-complétion par les choix d'écriture disponible de l'instruction en cours d'écriture.

➢ Il propose aussi une fenêtre de compilation ou s'affiche les éventuelles erreurs de compilation et une fenêtre d'exécution qui permet de visualiser les résultats de l'application.
Netbeans a l'avantage d'être distribué en open source et être entièrement gratuit, pour cette raison et pour sa facilité d'installation et d'usage sous Windows et les autres système comme Linux, nous l'avons choisi comme EDI durant l'implémentation de notre travail.

IV.4 Méthode de Benchmark

IV.4.1 Description

Une méthode de benchmark est un banc d'essai (test) permettant de mesurer les performances d'un système pour le comparer (logiciel, matériels, architectures,…) à d'autre système ou comparer des opérations dans le même système, c'est est un test dont l'objectif est de déterminer la performance d'un système informatique. L'acception la plus courante de ce terme (Benchmarks) est celle dans laquelle ses tests vont avoir pour objectif de mesurer les temps de l'implémentation et le niveau de sécurité …etc. Un Benchmark permet de valider correctement une application on un système avant son déploiement, tout en qualité de service qu'en consommation de ressources [27].

IV.4.2 Types de benchmark

Les Benchmarks peuvent être de plusieurs types, notamment :

➢ **Test de charge** : il s'agit d'un test permet de mettre en évidence les points sensibles et critiques de l'architecture technique. Il permet en outre de mesurer le dimensionnement des serveurs, de la bande passante nécessaire sur le réseau, etc.

➢ **Test de performance** : proche du Test de Charge, il s'agit d'un test au cours duquel on va mesurer les performances de l'application soumise à une charge d'utilisateurs. Les informations recueillies concernent les temps de réponse utilisateurs, les temps de réponse réseau et les temps de traitement d'une requête sur le(s) serveur(s). Ce test permet de vérifier les performances intrinsèques à différents niveaux de charge d'utilisateurs.

➢ **Test de dégradations des transactions** : il s'agit d'un test technique primordial au cours duquel on ne va simuler que l'activité transactionnelle d'un seul scénario fonctionnel parmi tous les scénarios du périmètre des tests, de manière à déterminer quelle charge le système est capable de supporter pour chaque scénario fonctionnel et d'isoler éventuellement les transactions qui dégradent le plus l'ensemble du système. Ce test.

➢ **Test de capacité, test de montée en charge :** il s'agit d'un test au cours duquel on va simuler un nombre d'utilisateurs sans cesse croissant (par paliers) de manière à déterminer quelle charge limite le système est capable de supporter.

- **Test de stress :** il s'agit d'un test au cours duquel on va simuler l'activité maximale attendue tous s en heures de pointe de l'application, pour voir comment le système réagit au maximum de l'activité des utilisateurs.
- **Test de robustesse, d'endurance, de fiabilité**: il s'agit de tests permet voir si le système testé est capable de supporter une activité intense sur une longue période sans dégradations des performances et des ressources applicatives ou système.
- **Test aux limites** : il s'agit d'un test au cours duquel on va simuler en général une activité bien supérieure à l'activité normale, pour voir comment le système réagit aux limites du modèle d'usage de l'application [28].

Il existe d'autres types de tests, plus ciblés et fonction des objectifs à atteindre dans la campagne de tests et il est coûteux de réaliser l'ensemble de ces tests pour une application donnée.

IV.4.3 Benchmarking

Maintenant nous avons utilisé les benchmarks pour mesurer les performances d'un système pour Comparer des temps de calcul des opérations cryptographiques (ECIES, ECDH et ECDSA) entre les courbes $GF_p \ et \ GF_{2^m}$ et en utilisant des courbes nommés par les groupes de standards de sécurité comme NIST et SECG.

Les deux principaux benchmarks de comparaison des algorithmes crypto sont la sécurité et l'efficacité. La sécurité est évaluée par rapport à la résistance aux attaques et elle est mesurée en nombre de bits utilisés dans la taille de clés symétriques et asymétriques. L'efficacité est la rapidité de calcul (la charge de calcul crypto.), la taille des clés (en nombre de bits) et la taille du message chiffré/signé à envoyer (la bande passante). Les opérations crypto se mesurent en général en milliseconde (ms).

Dans le monde de J2SE, les développeurs ont plusieurs outils pour examiner la performance du code et l'usage de la mémoire. Nous avons utilisé la méthode ancienne pour réaliser le benchmarking.

- **Utilisation de la mémoire**

Pour tester l'utilisation de la mémoire, on peut utiliser les méthodes de la classe java.lang.Runtime :

- ✓ public long freeMemory() : retourne la mémoire courante disponible en octets.
- ✓ public long totalMemory() : retourne la mémoire totale de l'environnement d'exécution en cours, en octets. Ce nombre peut changer parce que l'environnement hôte

(l'OS du dispositif) peut donner plus de mémoire à l'environnement d'exécution Java. Pour trouver la quantité mémoire utilisée par un objet, on peut procéder de cette manière :

```
Runtime runtime = Runtime.getRuntime();
long before, after;
System.gc();
before = runtime.freeMemory();
Object newObject = new String();
after = runtime.freeMemory();
long size = before - after;
```

- **Calcul des temps d'exécution**

Pour calculer le temps d'exécution, on utilise la méthode de la classe java.lang.System :

- ✓ public static long currentTimeMillis()

On peut calculer le temps d'exécution d'une méthode avec ce code :

```
long start, finish;
start = System.currentTimeMillis();
appel_Methode();
finish = System.currentTimeMillis();
long duration = finish - start;
```

Pour un temps d'exécution plus précis, on peut prendre plusieurs mesures et calculer une moyenne. Dans l'exemple suivant la procédure DigestSHA1() exécute 1e6 itérations de calcul de l'empreinte SHA1 :

```
start = System.currentTimeMillis();
DigestSHA1 ();  //1e6 itérations de calcul SHA1
finish = System.currentTimeMillis();
long duration = finish - start;
double timetaken = duration/1e6;
```

IV.5 Courbes elliptiques nommes (Named Curves)

Pour réaliser les benchmarks de comparaison nous avons utilisé les courbes elliptiques nommées de NIST et SECG, nous avons utilisé les trois algorithmes standards de cryptographie ECC déjà présentés dans le chapitre « courbes elliptiques et cryptographie » ces algorithmes sont : ECIES, ECDH et ECDSA.

Pour générer les paramètres de domaine, on peut procéder de cette manière :

```
/**Pour NIST**/
  X9ECParameters p1 = NISTNamedCurves.getByName("P-192");
  ECDomainParameters param1 = new ECDomainParameters(p1.getCurve(), p1.getG(),
  p1.getN(), p1.getH());
/**Pour SECG**/
  X9ECParameters p2 = SECNamedCurves.getByName("sect233k1");
  ECDomainParameters param2 = new ECDomainParameters(p2.getCurve(), p2.getG(),
   p2.getN(), p2.getH());
```

Également, on peut générer les paramètres des domaines dans courbe NIST ou SECG de cette manière :

```
ECGenParameterSpec ecSpec = new ECGenParameterSpec(“P-192”);
ECGenParameterSpec ecSpec = new ECGenParameterSpec(“secp160k1”);
```

IV.6 Résultat de benchmark et Comparaison entre les courbes (PF,BF)

Les algorithmes ECC utilisés dans les opérations de benchmark sont :

IV.6.1 Echange de clé : ECDH (Elliptic Curve Diffe Hellman)

L’opération ECDH comprend la génération de deux paires de clés (privé/publique) des entités A et B et comparaison du secret DH partagé calculé par les deux entités si (A et B génèrent le même secret) alors opération ECDH réussie sinon opération ECDH échouée. La comparaison utilise ECDH avec SHA1 et ECDH avec SHA256 (Table IV.1, Table IV.2). L’interface réalisant ce benchmark ECDH se présente comme suit:

Figure IV.1 : Interface De Benchmark ECDH

1. choisir la courbe nommée
2. choisir la fonction de hachage et clique sur générer.

Les tableaux suivants montrent les temps d'exécution des opérations cryptographiques ECDH sur les courbes EC(PF) et EC(BF) dans NIST et SECG

Table IV.1: ECDH avec les courbes PF, BF de NIST

Opérations cryptographiques ECC (en ms)	P-192	B-163	P-224	B-233	P-256	B-283	P-384	B-409	P-521	B-571
Génération 2 paires de clés	53,543	73,224	73,685	163,616	104,863	253,345	264,951	585,678	582,768	1293,09
Génération 2 paires de clés avec SHA1	53,664	69,671	74,038	153,58	101,853	233,059	264,686	574,356	585,84	1296,886
Génération 2 paires de clés avec SHA256	54,195	70,371	74,428	154,141	102,554	234,164	267,572	578,309	588,805	1296,888

Table IV.2 : ECDH avec les courbes PF, BF de SECG

Opérations cryptographiques ECC (en ms)	secp160k1	sect163k1	secp160r1	sect163r1	secp160r2	sect163r2	secp192r1	sect193r1	secp224k1	sect233k1
Génération 2 paires de clés	35,705	68,573	36,081	69,497	35,947	68,030	53,926	100,820	74,773	148,062
Génération 2 paires de clés avec SHA1	35,825	68,266	36,077	67,329	36,087	68,486	53,63	97,134	74,489	150,213
Génération 2 paires de clés avec SHA256	35,908	72,452	36,679	67,862	36,623	70,885	56,513	97,319	79,711	150,737

- ✓ Après l'application de l'algorithme ECDH, nous constatons que le temps de génération de deux paires de clés avec SHA256 est plus lent par rapport à la génération de deux paires de clés avec SHA1 car la fonction de hachage SHA256 produit une empreinte de 256 bits donc elle prend plus de temps, tandis que la fonction SHA1 produit une empreinte de 160 bits.
- ➢ Le résultat de ce benchmark montre que l'opération ECDH est plus lente avec les courbes BF par rapport aux courbes PF. Ceci est dû au choix de la représentation utilisée pour définir la courbe, dans le cas de courbe PF les opérations arithmétiques comme l'addition, et les paramètres de domaines prennent des valeurs dans un ensemble polynomiale mais dans le cas d'une courbe BF ces paramètres prennent des valeurs dans l'ensemble [0, p-1]. En plus toutes les opérations sont calculées modulo p (p premier) et le corps $\mathbb{F}_{2^n}$ est considéré comme trop structuré.

IV.6.2 Cryptage : ECIES (Elliptic Curve Integration Encryption Standard)

Dans ECIES, un secret partagé de type Diffie-Hellman est utilisé pour dériver deux clés symétriques k1 et k2, La clé k1 est utilisée pour chiffrer le texte clair en utilisant un

algorithme de chiffrement symétrique(comme AES), k2 est utilisé pour authentifier le texte chiffré résultant.

pour chiffrer un message clair m , on peut procéder de cette manière :

Pour chiffrement par bloc en mode CBC

```
BufferedBlockCipher c = new PaddedBufferedBlockCipher(
                                    new CBCBlockCipher(new AESEngine()));
IESEngine i1 = new IESEngine(new ECDHBasicAgreement(),new
KDF2BytesGenerator(new SHA1Digest()),new HMac(new SHA1Digest()), c);
```

Pour chiffrement par flot

```
IESEngine i1 = new IESEngine(new ECDHBasicAgreement(),new
KDF2BytesGenerator(new SHA1Digest()),new HMac(new SHA1Digest()));
```

On va comparer les temps de calcul pour la génération de deux paires de clés, chiffrement et déchiffrement. Les tableaux suivants montrent les temps d'exécution des opérations cryptographiques ECIES sur les courbes EC(PF) et EC(BF) dans NIST et SECG :

Table IV.3 : ECIES avec les courbes PF, BF dans NIST

Opérations cryptographiques ECC (en ms)	P-192	B-163	P-224	B-233	P-256	B-283	P-384	B-409	P-521	B-571
Génération 2 paires de clés	26,522	33,570	37,207	74,799	50,798	114,265	134,069	282,848	298,968	635,807
Chiffrement stream cipher	13,120	16,717	18,531	37,422	25,350	55,330	66,651	138,280	143,768	317,853
Déchiffrement stream cipher	13,064	16.664.	18,447	37,405	25,268	55,293	66,458	138,270	143,663	317,424
Chiffrement Block cipher AES-CBC	13,242	17.51.	18,705	37,578	25,486	55,639	65,888	139,934	147,632	319,465
Déchiffrement Block cipher AES-CBC	13,081	17,407	18,505	37,519	25,446	55,443	65,653	140,079	147,588	319,883

Table IV.4 : ECIES avec les courbes PF, BF dans SEC

Opérations cryptographiques ECC (en ms)	secp160k1	sect163k1	secp160r1	sect163r1	secp160r2	sect163r2	secp192r1	sect193r1	secp224k1	sect233k1
Génération 2 paires de clés	17,522	34,389	17,619	33,430	17,590	33,532	26,312	48,450	37,043	73,754
Chiffrement stream cipher	8,685	17,134	8,765	16,651	8,732	16,942	13,109	24,012	18,223	36,797
Déchiffrement stream cipher	8,673	17,140	8,746	16,645	8,705	16,747	12,986	23,866	18,188	36,728
Chiffrement Block cipher AES-CBC	9,146	17,774	9,342	17,394	9,121	17,053	17,469	24,723	18,828	37,197
Déchiffrement Block cipherAES-CBC	9,138	17,764	9,295	17,367	9,110	16,980	17,460	24,671	18,786	37,095

- Après l'application de l'algorithme ECIES, nous remarquons que le temps de chiffrement et déchiffrement par bloc est plus grand a ceux par flot parce que dans le cas du chiffrement et déchiffrement par bloc, chaque bloc dépend du bloc précédent, et le chiffrement à l'intérieur du bloc utilise le mécanisme de chiffrement par flot, mais dans le du chiffrement par flot chaque bit est chiffré indépendamment des autres.
- Le même remarque sur le temps d'exécution des opérations ECDH s'applique aussi pout les opérations crypto basées ECIES, les courbes PF sont plus performants que les courbes BF.

IV.6.3 Signature numérique :ECDSA (Elliptic Curve digital signature Algorithm)

L'interface réalisant ce benchmark ECDSA se présente comme suit:

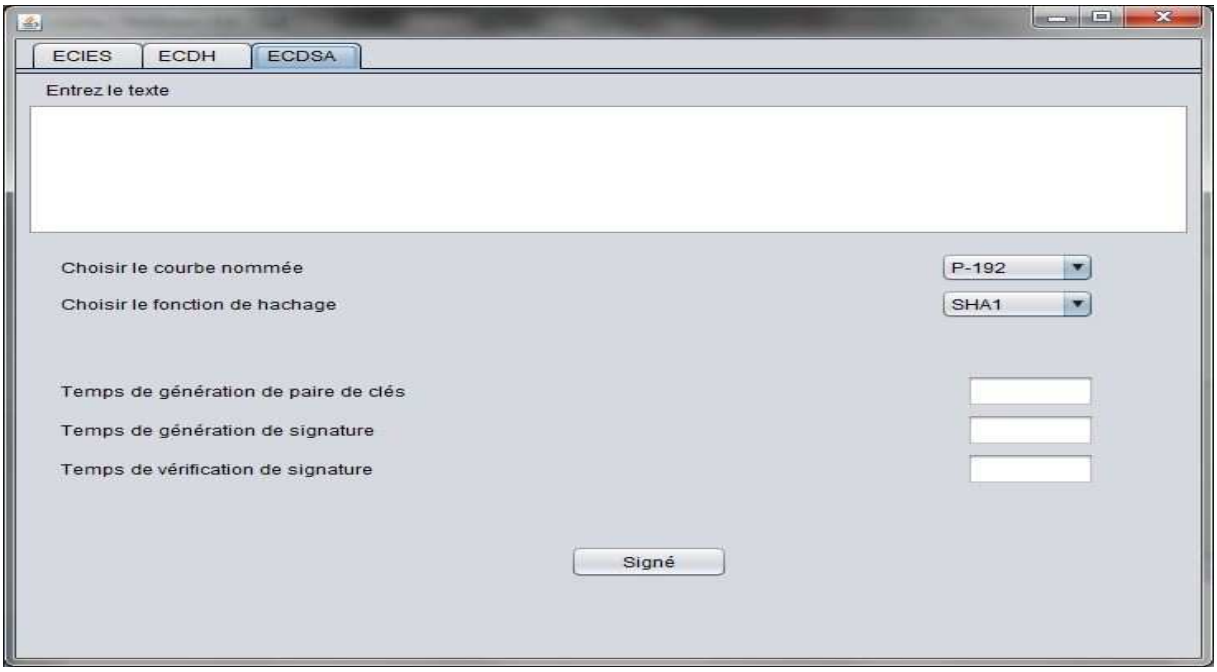


Figure IV.2 : Interface De Benchmark ECDSA

1. Entrer un message
2. Choisir la courbe nommée
3. Choisir la fonction de hachage et clique signé

Les tableaux suivants (Table IV.5, Table IV.6), donne une comparaison des temps de calcul en millisecondes pour une génération de paire de clés, une génération de signature et sa vérification en utilisant le schéma ECDSA, avec des courbes nommées *EC(PF)* et *EC(BF)* de NIST et SECG :

Table IV.5 : ECDSA avec les courbes PF, BF dans NIST

Opérations cryptographiques ECC (en ms)	P-192	B-163	P-224	B-233	P-256	B-283	P-384	B-409	P-521	B-571
Génération un seul paire de clés	13,513	17,432	18,830	37,722	26,134	56,620	68,222	138,423	147,398	317,168
Génération de signature avec SHA1	13,378	17,462	18,777	37,709	25,905	56,668	68,017	138,562	147,395	317,276
vérification de signature avec SHA1	17,17	22,556	24,077	49,143	33,509	73,904	88,327	180,677	190,936	415,284
Génération de signature avec SHA256	13,581	17,687	19,588	38,286	25,921	56,864	68,33	140,39	147,587	321,506
vérification de signature avec SHA256	17,195	22,727	25,191	49,977	34,067	74,337	88,776	182,963	191,314	421,455

Table IV.6 : ECDSA avec les courbes PF, BF dans SECG

Opérations cryptographiques ECC (en ms)	secp160k1	sect163k1	secp160r1	sect163r1	secp160r2	sect163r2	secp192r1	sect193r1	secp224k1	sect233k1
Génération un seul paire de clés	9,102	17,478	9,201	17,304	9,816	17,347	13,830	24,685	18,842	37,98
Génération de signature avec SHA1	9,07	17,542	9,182	17,161	9,599	17,354	13,869	24,756	18,788	38,139
vérification de signature avec SHA1	11,511	22,622	11,646	22,31	12,329	22,437	17,548	31,976	24,198	49,579
Génération de signature avec SHA256	9,178	19,705	9,851	17,862	9,759	18,788	13,92	24,812	19,27	40,7
vérification de signature avec SHA256	11,719	25,286	12,839	23,177	12,339	24,299	17,722	32,184	24,662	52,735

- ✓ Après l'application de l'algorithme ECDSA, il nous apparaît que la génération de signature avec SHA256 est plus lente par rapport génération de signature avec SHA1, l'empreinte produite avec SHA256 prend plus de temps, sa taille tient sur plus de bits.
- ✓ Les résultats font ressortir que dans l'algorithme ECDSA, le temps de vérification de signature est plus long mais le temps de génération de signatures est plus rapide puisque le destinataire peut vérifier que le message n'a pas été altéré durant la communication. Lors de la réception du message, il suffit au destinataire de calculer le haché du message reçu et de le comparer avec le haché accompagnant le document. Si le message (ou le haché) a été falsifié durant la communication, les deux empreintes ne correspondront pas.
- ✓ La réalisation des benchmarks pour mesurer les opérations crypto basées ECC dans différentes familles (courbes PF, BF) font ressortir que la performance de courbes PF est meilleure que les courbes BF dans les machines ayant des processeurs généralistes, le calcul des opérations arithmétique (addition, multiplication,…) sur BF est plus complexe par rapport PF.
  - ✓ La Figure IV.3 montrelacomparaison de performance d'opération crypto basée ECC entre deux courbes nommées dans NIST sur plates-formes (Processor: Intel(R) Core (TM) 2 Duo CPU 2.20GHz, Memory: 3072MB RAM).

La Figure IV.3 montre lacomparaison de performance d'opération crypto basée ECC entre deux courbes nommées dans NIST sur plates-formes (Processor: Intel(R) Core (TM) 2 Duo CPU 2.20GHz, Memory: 3072MB RAM).

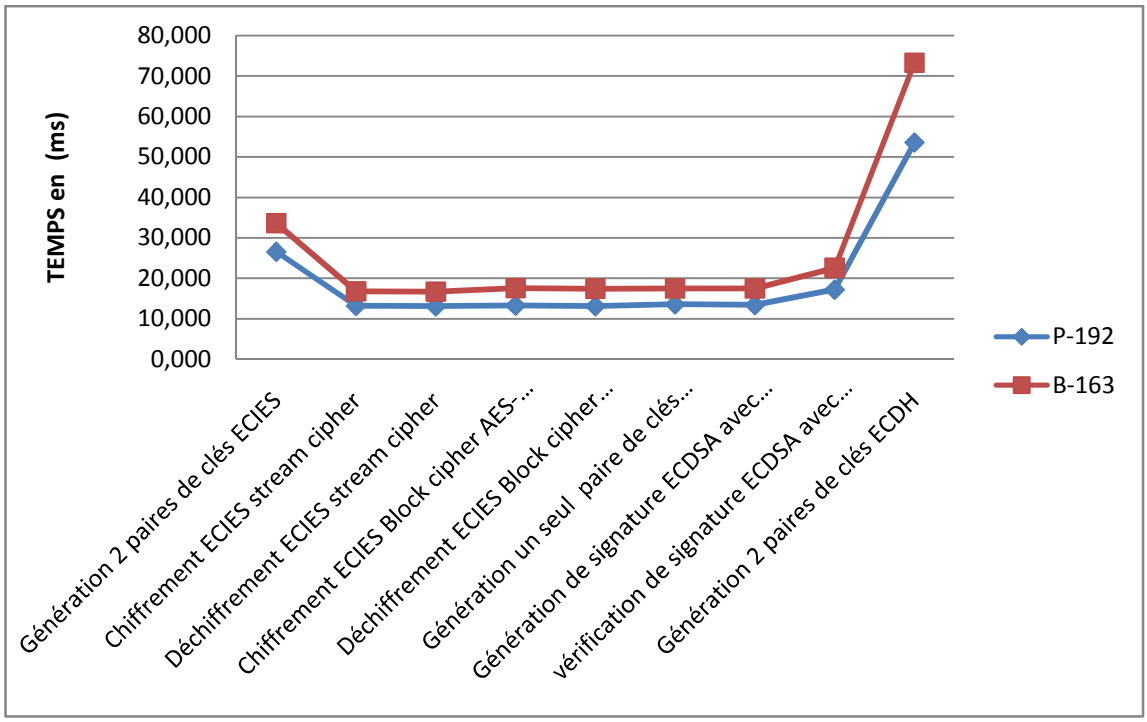


Figure IV.3 : Comparaison performance des opérations cryptographiques

IV.7 Conclusion

Nous avons réalisé ces opérations ECC sur notre lap top PC (Processor: Intel(R) Core (TM) 2 Duo CPU 2.20GHz, Memory: 3072MB RAM) en utilisant la plateforme J2SE et le provider JCE de BouncyCastle, et nous avons trouvé que les courbes elliptiques sur PF exécutent mieux les opérations cryptographiques que celles sur BF.

Les courbes sur PF sont appropriées aux implémentations logicielles sur un PC, et tirent profil (comme RSA) de l'arithmétique entière des processeurs généralistes. Tandis que les courbes sur BF sont appropriées aux implémentations matérielles personnalisées, leur calcul arithmétique nécessite peu de circuits logiques. Dans une implémentation matérielle personnalisée (comme des dispositifs contraints), les courbes BF réalisent un coût plus réduit que leurs homologues PF.

Conclusion Générale

Les crypto systèmes basés sur les courbes elliptiques se posent en alternative efficace face à l'incontournable RSA. En effet, ils exploitent un problème mathématique différent, qui est réputé pour sa solidité égale à RSA pour des clés de longueur bien inférieure. Cela les rend parfaitement adaptés aux utilisations embarquées, comme les cartes à puce par exemple, où la mémoire et la puissance des processeurs ne sont pas suffisants pour réaliser en un temps convenable les calculs exigés par RSA, DH ou DSA.

De nos jours la cryptographie est en perpétuelle évolution afin de pouvoir répondre aux besoins de sécurisation des données qui ne cessent d'augmenter. En effet, les crypto systèmes se doivent d'être performants face à des attaques de plus en plus nombreuses. C'est pourquoi nous ne pouvons pas prédire combien de temps les crypto systèmes basés sur les courbes elliptiques seront les plus efficaces au niveau sécurité.

Les crypto systèmes basés sur les courbes elliptiques sont :

- Plus rapide et utilisation mémoire moindre
- Utilisation pour systèmes embarqués
- plus efficaces au niveau sécurité

Les benchmarks de comparaison réalisés pour les opérations crypto basée ECC entre différentes familles de courbes (PF, BF), montrent que la performance de courbes PF est meilleure par rapport leurs homologues BF dans les machines utilisant des processeurs généralistes comme les PCs. les courbes PF sont les plus utilisées et elles ont un certain avantage lorsqu'on considère l'interopérabilité. Certains experts considèrent aussi que les courbes PF sont plus résistants aux attaques.

Bibliographie

[1]. *http://www.apprendre-en-ligne.net/crypto/histoire*. [En ligne] [Citation : 09 03 2013]

[2]. *http://math.univ-lyon1.fr/~roblot/masterpro.html /chapitre1*. [En ligne] [Citation : 06 11 2012]

[3]. Barsky, Daniel. Cours de Cryptographie. février 2006.

[4]. Duquesne, S. Cryptographie sur les courbes elliptiques. avril 2005.

[5]. Omar Cheikhrouhou, Les Algorithmes Cryptographiques Symétriques , 2010-2011

[6]. Bayart, Frederic. La saga du DES. *http://www.bibmath.net/crypto* . [En ligne] [Citation : 09 03 1013]

[7]. SecuriteInfo.com,L'AES : Advanced Encryption Standard. *http://www.securiteinfo.com/cryptographie/aes.shtml.* [En ligne] [Citation : 11 03 2013]

[8]. Lo, Johnny Li-Chang. A framework for cryptography algorithms on mobile devices. s.l. : Faculty of Engineering, Build and Information Technology Department of Computer Science UNIVERSITY OF PRETORIA, 2007.

[9]. jean-Bruno, Emonet. Algorithmes de chiffrement. 22 juin 2005.

[10]. Mécanismes cryptographiques. Paris : Direction centrale de la sécurité des systèmes d'information, 19 décembre 2006.

[11]. Nitaj, Abderrahmane. CRYPTANALYSE DE RSA. Laboratoire de Mathématiques Nicolas Oresme, Université de Caen, France : s.n., 28 juin 2009.

[12]. National Institute of Standards and Technology (NIST), Secure Hash Standard. Federal Information Processing Standards Publication 180-2, 2002, en ligne *http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.*

[13]. Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu, Finding Collisions in the Full SHA-1, 2005, en ligne *http://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf.*

[14]. Niels Ferguson, Bruce Schneier. Practical Cryptography. Indiana: Wiley Publishing, Inc. : s.n., 2003.

[15]. Stephen A. Thomas, SSL & TLS Essentials Securing the web, Appendix A: X.509 Certificates, Wiley Computer Publishing, John Wiley & Sons, Inc., 2000. .

[16]. Introduction à la cryptographie, Hervé Schauer Consultants.

[17]. RSA Laboratories, PKCS#1 v2.1: RSA Cryptography Standard, PKCS Standards. 2002.

[18]. Darrel Hankerson, Alfred Menezes et Scott Vanstone. Guide to Elliptic Curve Cryptography. 2004.

[19]. Bonecaze, Stéphane Ballet et Alexis. Courbes Elliptiques Application à la Cryptographie . s.l. : Ecole d'Ingénieur de Luminy., 2011-2012.

[20]. Duquesne, S. Cryptographie sur les courbes elliptiques. s.l. : ,Ecole Jeunes Cherceurs, 5 avril 2005.

[21]. Dumont, R. Le chiffrement par clé publique.

[22]. On the Web sun.com, White Paper Java Security Overview, Sun Microsystems. April 2005.

[23]. Oracle Java SE Documetantation, Java ™ Cryptography Architecture (JCA) Reference Guide for JavaTM Platform Standard Edition 6 .

[24]. Hook, David. Beginning Cryptography with Java (Chapter 1 :The JCA and the JCE) . August2005.

[25]. The Legion of the Bouncy Castle, SPECIFICATIONS. *http://www.bouncycastle.org/specifications.html. [En ligne]*

[26]. *http://www.developpez.net/forums/d11833/general-java/conceptdsharp*. [En ligne] [Citation : 05 06 2013]

[27]. *http://www-igm.univ-mlv.fr/~dr/XPOSE2008/Benchmarking et optimisations en Java .* [En ligne] [Citation : 05 06 2013]

[28]. *http://fr.wikipedia.org/wiki/Test_de_performance - Wikipédia* . [En ligne] [Citation : 05 06 2013]

Résumé : La cryptographie à clé publique a réalisé un succès énorme dans la protection des transactions web, les échanges de mails et les sessions. Comme les ressources de calcul disponible au niveau des cryptanalyses augmentent et leur technique de cryptanalyse s'améliorent, le niveau de sécurité de l'algorithme crypto doit augmenter aussi avec le temps. Les architectes de sécurité font face à ce défi, en spécifiant des tailles de clés plus grandes. Cependant, ceci a un impact sur la performance de calcul de ces algorithmes, cet impact n'est pas le même pour chaque algorithme clé publique. RSA qui est le système clé publique le plus largement utilisé, a des problèmes sérieux avec des tailles de clés très grandes. Pour cette raison il est important d'aller vers d'autres types d'algorithmes offrant des caractéristiques de sécurité équivalentes ou meilleures mais avec des tailles de clé plus réduites. Ces algorithmes sont basés sur une arithmétique utilisant des courbes elliptiques. Décrite initialement en 1985. Ces systèmes représente un avantage pour les cartes à puces dont l'espace mémoire est très limité. Notre travail consiste à réaliser des benchmarks pour mesurer les opérations crypto basées ECC, en choisissant des courbes elliptiques appartenant aux différentes familles (courbes PF, BF), pour les comparer entre eux. Le système réalisant ces benchmarks, peut être implémenté, testé et validé par des outils et des packages crypto open source (comme BouncyCastle) sous la plateforme J2SE.

Mots clefs : ECC, courbes PF et BF, crypto système, clés, sécurité, cryptographie dans Java, provider JCE, BouncyCastle, la plateforme J2SE.

Abstract: Public key cryptography realized an enormous success in the protection of web transactions, mails and sessions exchanging. Since the resources of computing available to cryptanalysts have grown, and their techniques are more and more improved, The security level must grow equally, security architects facing this challenge, produced bigger key sizes, however, this has impacted the performance of that algorithms. RSA the largest used public key system, has serious problems with key sizes,. For this end, it is important to search for other algorithms, providing an equivalent or better security level, and at the same time use more reduced key sizes. Elliptic curves cryptography provide such a good alternative. designed initially in 1985, this system represents advantages for smartcards with limited memory space. In our work we realize benchmarks for measuring operations in ECC based cryptography, by choosing elliptic curves belonging to different families (PF, BF curves), to compare between them. The system releasing these benchmarks can be implemented, tested , and validated using open sources tools and packages (as BouncyCastle) under J2SE platform.

Keywords: ECC, curves PF and BF, cryptosystem, keys, security, cryptography in Java, JCE Provider, Bouncy Castle, the J2SE platform.