# Abstract

In this thesis, we propose a variant of the ANSI X9.62 ECDSA. We give a brief introduction to the digital signature algorithm in chapter 2, and then give the basic concepts of the elliptic curve cryptosystems in chapter 3. The next chapter includes the elliptic curve version of DSA, and finally a variant of ECDSA will be given in chapter 5.

Keywords: digital signature algorithm(DSA), elliptic curve digital signature algorithm(ECDSA).

# 摘 要

本篇論文主要目的在探討橢圓曲線版的數位簽署。第二章我們介紹數位簽署的密碼系統 且在第三章簡介橢圓曲線密碼系統的基本概念。緊接著介紹橢圓曲線版的數位簽署(*ECDSA*)以及我們所提出的一個改良的版本。

關鍵字： 數位簽署、 橢圓曲線數位簽署。

# Contents

# Chapter 1

# Introduction

The Digital Signature Algorithm was specified in a U.S. Government Federal Information Processing Standard called the Digital Signature Standard. Its security is based on the computational intractability of the discrete logarithm problem in prime-order subgroups of $(\mathbb{Z}/p\mathbb{Z})^{\times}$.

Elliptic curve cryptosystems were invented by Neal Koblitz and Victor Miller in 1985. They can be viewed as elliptic curve analogues of the older discrete logarithm cryptosystems in which the subgroup of $(\mathbb{Z}/p\mathbb{Z})^{\times}$ is replaced by the group of points on an elliptic curve over a finite field. The mathematical basis for the security of elliptic curve cryptosystems is the computational intractability of the elliptic curve discrete logarithm problem.

Since the Elliptic Curve Digital Logarithm Problem appears to be significantly harder than the Digital Logarithm Problem, the strength-per-key-bit is substantially greater in elliptic curve systems than in conventional discrete logarithm systems. Thus, smaller parameters can ce used in Elliptic Curve Cryptosystem than with Digital Logarithm systems but with equivalent levels of security. The advantages that can be gained from smaller parameters include speed (faster computations) and smaller keys and certificates. These advantages are especially important in environments where processing power, strong space, bandwidth, or power consumption is constrained.

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm. ECDSA was first proposed in 1992 by Scott Vanstone in response of NIST (Nation Institute of Standards and Technology) request for public comments on their proposal for Digital Signature Schemes. It was accepted in 1998 as an ISO (International Standards Organization) standard (ISO 14888-3), accepted in 1999 as an ANSI (American National Standards Institute) standard (ANSI X9.62), and accepted in 2000 as an IEEE (Institute of Electrical and Electronics Engineers) standard (IEEE 1363-2000) and FIPS standards (FIPS 186-2). It is also under consideration for inclusion in some other ISO standards. In this thesis, we propose a variant of the ANSI X9.62 ECDSA.

We give a brief introduction of the digital signature algorithm in chapter 2, and then give the basic concepts of the elliptic curve cryptosystems in chapter 3. The next chapter comes the elliptic curve version of DSA and finally a variant of ECDSA will be given in chapter 5.

# Chapter 2

# Digital Signature Algorithm

## 2.1 Discrete Logarithm Problems

Fix a prime $p$. Let $\alpha$ and $\beta$ be two nonzero integers mod $p$ and suppose

$$\beta \equiv \alpha^x \pmod{p}.$$

The problem of finding $x$ is called the discrete logarithm problem (DLP). It is easy to compute $\alpha^x \bmod p$, but solving $\alpha^x \equiv \beta$ for $x$ is probably hard.

## 2.2 The Digital Signature Schemes

The conventional handwritten signature on a document is used to certify that the signer is responsible for the content of the document. The signature is physically a part of the document and while forgery is certainly possible, it is difficult to do so convincingly. Trying to mimic a handwritten signature in a digital medium leads to a difficulty since cut and paste operations can be used to create a perfect forgery. Thus, we need to have a way of signing messages digitally which is functionally equivalent to a physical signature, but which is at least as resistant to forgery as its physical counterpart.

Schemes which provide this functionality are called Digital Signature Schemes. A Digital Signature Scheme will have two components, a private signing algorithm which permits a user to securely sign a message and a public verification algorithm which permits anyone to verify that the signature is authentic. The signing algorithm needs to "bind" a signature to a message in such a way that the signature can not be pulled out and used to sign another document, or have the original message modified and the signature remain valid. For practical reasons it would be necessary for both algorithms to be relatively fast and if small computers such as smart cards are to be used, the algorithms can not be too computationally complex.

## 2.3 Hash Function

A hash function $H$ is a transformation that takes an input $m$ and returns a fixed-size string, which is called the hash value $h$ (that is, $h = H(m)$). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography, the hash functions are usually chosen to have some additional properties.

The basic requirements for a cryptographic hash function are as follows.

- The input can be of any length.
- The output has a fixed length.
- $H(m)$ is relatively easy to compute for any given $m$.
- $H(m)$ is one-way.
- $H(m)$ is collision-free.

A hash function $H$ is said to be *one-way* if it is hard to invert, where "hard to invert" means that given a hash value $h$, it is computationally infeasible to find some input $m_1$ such that $H(m_1) = h$.

If, given a message $m_1$, it is computationally infeasible to find a message $m_2$ not equal to $m_1$ such that $H(m_1) = H(m_2)$, then $H$ is said to be a weakly collision-free hash function.

A strongly collision-free hash function $H$ is one for which it is computationally infeasible to find any two messages $m_1$ and $m_2$ such that $H(m_1) = H(m_2)$

The hash value represents concisely the longer message or document from which it was computed; this value is called the message digest. One can think of a message digest as a "digital fingerprint" of the larger document. Examples of well known hash functions are MD2 and MD5 and SHA .

## 2.4 The Digital Signature Algorithm (DSA)

The DSA was proposed in August 1991 by U.S. National Institute of Standards and Technology (NIST) and was specified in a U.S. Government Federal Information Processing Standard (FIPS 186) called the *Digital Signature Standard* (DSS). The DSA can be viewed as a variant of the ElGamal signature scheme. Its security is based on the intractability of the discrete logrithm problem in prime-order subgroup of $(\mathbb{Z}/p\mathbb{Z})^\times$.

- DSA DOMAIN PARAMETER GENERATION

  Domain parameter are generated for each entity in a particular security domain.

  1. Select a 160-bit prime $q$ and a 1024-bit prime $p$ with the property that $q|p-1$.
  2. Select an element $h \in (\mathbb{Z}/p\mathbb{Z})^\times$ and compute $g = h^{(p-1)/q} \pmod{p}$. (Repeat until $g \neq 1$.)
  3. Domain parameters are $p, q$ and $g$.

- DSA KEY PAIR GENERATION

  Each entity A in the domain with domain parameters $(p, q, g)$ does the following:

  1. Select a random or pseudo-random integer $x$ such that $1 \leq x \leq q-1$.
  2. Compute $y = g^x \pmod{p}$.
  3. A's public key is $y$; A's private key is $x$.

- DSA SIGNATURE GENERATION

  To sign a message $m$, A does the following:

  1. Select a random or pseudo-random integer $k$ such that $1 \leq k \leq q-1$.
  2. Compute $X = g^k \pmod{p}$ and $r = X \pmod{q}$. If $r = 0$ thaen go to step 1.
  3. Compute $k^{-1} \pmod{q}$.
  4. Compute $H(m) = \text{SHA-1}(m)$.

5. Compute $s = k^{-1}(H(m) + xr) \pmod{q}$. If $s = 0$ then go to step 1.
6. A's signature for the message $m$ is $(r, s)$.

- DSA SIGNATURE VERIFICATION

  To verify A's signature $(r, s)$ on $m$, B obtains authentic copies of A's domain parameters $(p, q, g)$ and public key $y$ and does the following:

  1. Verify that $r$ and $s$ are integers in the interval $[1, q-1]$.
  2. Compute $H(m) = \text{SHA-1}(m)$.
  3. Compute $w = s^{-1} \pmod{p}$.
  4. Compute $u_1 = H(m)w \pmod{q}$ and $u_2 = rw \pmod{q}$.
  5. Compute $X = g^{u_1}y^{u_2} \pmod{p}$ and $v = X \pmod{q}$.
  6. Accept the signature if and only if $v = r$.

- PROOF THAT SIGNATURE VERIFICATION WORKS

  If a signature $(r, s)$ on a message $m$ was indeed generated by A, then $s = k^{-1}(H(m) + xr) \pmod{q}$. Rearranging gives

$$\begin{aligned} g^k &= g^{s^{-1}(H(m)+xr)} \\ &= g^{s^{-1}H(m)+s^{-1}rx} \\ &= g^{H(m)w}y^{rw} \\ &= g^{u_1}y^{u_2}. \end{aligned}$$

# Chapter 3

# Elliptic Curves Cryptosystems

## 3.1 Elliptic Curves over Finite Field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$

Let $p > 3$ be an odd prime. An elliptic curve $E$ over $\mathbb{F}_p$ is defined by an equation of the form

$$y^2 = x^3 + ax + b, \tag{3.1}$$

where $a, b \in \mathbb{F}_p$, and $4a^3 + 27b^2 \not\equiv 0 \bmod p$. The set $E(\mathbb{F}_p)$ consists of all points $(x, y)$, $x \in \mathbb{F}_p$, $y \in \mathbb{F}_p$ that satisfy the defining equation(3.1), together with a special point $\mathcal{O}$ called the point at infinity.

**Example:** (Elliptic curve over $\mathbb{F}_{23}$) Let $p = 23$ and consider the elliptic curve $E : y^2 = x^3 + x + 4$ defined over $\mathbb{F}_{23}$. Note that $4a^3 + 27b^2 = 4 + 432 = 436 \equiv 22 \mod 23$, so $E$ is indeed an elliptic curve. The points in $E(\mathbb{F}_{23})$ are $\mathcal{O}$ and the following:

$$\begin{array}{lllll}
(0,2), & (0,21), & (1,11), & (1,12,) & (4,7), \\
(4,16), & (7,3), & (7,20), & (8,8), & (8,15), \\
(9,11), & (9,12), & (10,5), & (10,18), & (11,9), \\
(11,14), & (13,11), & (13,12), & (14,5), & (14,18), \\
(15,6), & (15,17), & (17,9), & (17,14), & (18,9), \\
(18,14), & (22,5), & (22,19). & &
\end{array}$$

Addition formula: There is a rule, called the *chord-and-tangent rule*, for adding two points on an elliptic curve $E(\mathbb{F}_p)$ to give a third elliptic curve point. Together with this addition operation, the set of points $E(\mathbb{F}_p)$ forms a group with $\mathcal{O}$ serving as its identity. It is this group that is used in the construction of elliptic curve cryptosystems.

The addition rule is best explained geometrically. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points on an elliptic curve $E$. Then the *sum* of $P$ and $Q$, denoted $R = (x_3, y_3)$, is defined as follows. First draw the line through $P$ and $Q$; this line intersects the elliptic curve in a third point. Then $R$ is the reflection of this point in the $x$-axis. This is depicted in Figure 3.1. The elliptic curve in the figure consists of two parts, the ellipse-like figure and the infinite curve.

If $P = (x_1, y_1)$, then the *double* of $P$, denoted $R = (x_3, y_3)$, is defined as follows. First draw the tangent line to the elliptic curve at $P$. This line intersects the elliptic curve in a second point. Then $R$ is the reflection of this point in the $x$-axis. This is depicted in Figure 3.2.

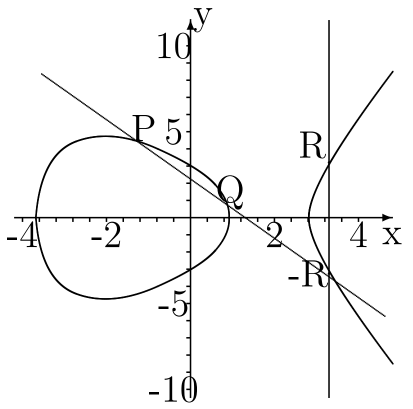

Figure 3.1: $P + Q = R$

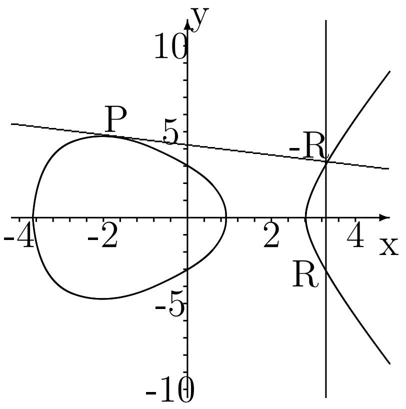

Figure 3.2: $P + P = R$

The following algebraic formulae for the sum of two points and the double of a point can be derived from the geometric description.

1. Let $P = (x_1, y_1) \in E(\mathbb{F}_p)$.Then $P + \mathcal{O} = \mathcal{O} + P = P$ for all $p \in E(\mathbb{F}_p)$.

2. If $P = (x_1, y_1) \in E(\mathbb{F}_p)$, then $(x_1, y_1) + (x_1, -y_1) = \mathcal{O}$. (The point $(x_1, -y_1)$ is denoted by $-P$, and is called the *negative* of $P$; observe that $-P$ is indeed a point on the curve.)

3. (Addition Formula) Let $P = (x_1, y_1),\ Q = (x_2, y_2) \in E(\mathbb{F}_p)$.. Then $P + Q = (x_3, y_3)$ and the formula of $x_3$ and $y_4$ are

$$\begin{aligned} x_3 &= m^2 - x_1 - x_2 \\ y_3 &= m(x_1 - x_3) - y_1. \end{aligned}$$

where

$$m = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if P} \neq \text{Q} \\[2ex] \dfrac{3x_1^2 + a}{2y_1}, & \text{if P = Q} \end{cases}$$

**Example:** Let $p = 23$ and consider the elliptic curve $E : y^2 = x^3 + x + 4$ defined over $\mathbb{F}_{23}$.

1. (Point addition) Let $P = (4, 7),\ Q = (13, 11) \in E(\mathbb{F}_{23})$.Then $P + Q = (x_3, y_3)$ is computed as follows:

$$\begin{aligned} x_3 &= (\frac{11-7}{13-4})^2 - 4 - 13 = -8 \equiv 23 \pmod{23}. \\ y_3 &= (\frac{11-7}{13-4})(4-15) - 7 = -40 \equiv 6 \pmod{23}. \end{aligned}$$

   Hence $P + Q = (15, 6)$.

2. (Point doubling) Let $P = (4, 7)$. Then $2P = P + P = (x_4, y_4)$ is computed as follows :

$$\begin{aligned} x_4 &= (\frac{3(4)^2+1}{14})^2 - 8 = 217 \equiv 10 \pmod{23}. \\ y_4 &= (\frac{3(4)^2+1}{14})(4-10) - 7 = -97 \equiv 18 \pmod{23}. \end{aligned}$$

   Hence $2P = (10, 18)$.

## 3.2 Elliptic Curve Cryptosystems

Several approaches to encryption (or decryption) using elliptic curves have been analyzed. This paper describes one of them. The first task in this system is to encode the plaintext message $m$ to be sent as an $x$-$y$ point $P_m$. It is the point $P_m$ that will be encrypted as a cipher text and subsequently decrypted. Note that we cannot simply encode the message as the $x$ or $y$ coordinate of a point, because not all such coordinates are in $E(\text{mod}p)$. There are approaches to encoding. We developed a scheme that will be reported elsewhere. As with the key exchange system, an encryption (or decryption) system requires a point $G$ and an elliptic group $E(\text{mod}p)$ as parameters. Each user A selects a private key $\alpha_A$ and generates a public key

$$P_A = \alpha_A G.$$

To encrypt and send a message $P_m$ to B, A chooses a random positive integer $x$ and produces the cipher text $C_m$ consisting to the pair of points

$$C_m = \{xG, P_m + xP_B\}.$$

Note that A has used B's public key $P_B$. To decrypt the cipher text, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point:

$$P_m + xP_B - n_B(xG) = P_m + x(n_B G) - n_B(xG) = P_m.$$

A has masked the message $P_m$ by adding $xP_B$ to it.

Nobody but A knows the value of $x$, so even though $P_B$ is a public key, nobody can remove the mask $xP_B$. However, A also includes a "clue," which is enough to remove the mask if one knows the private key $n_B$. For an attacker to recover the message, the attacker would have to compute $x$ given $G$ and $xG$, which is hard.

# Chapter 4

# Elliptic Curve DSA (ECDSA)

## 4.1 Elliptic Curve DLP (ECDLP)

The elliptic curve discrete logarithm problem is the cornerstone of much of present-day elliptic curve cryptography. It relies on the natural group law on a non-singular elliptic curve which allows one to add points on the curve together. Given an elliptic curve $E$ over a finite field $\mathbb{F}$, a point on that curve, $P$, and another point you know to be an integer multiple of that point $Q$, the "problem" is to find the integer $n$ such that $nP = Q$.

The problem is computationally difficult unless the curve has a "bad" number of points over the given field, where the term "bad" encompasses various collections of numbers of points which make the elliptic curve discrete logarithm problem breakable. For example, if the number of points on $E$ over $\mathbb{F}$ is the same as the number of elements of $\mathbb{F}$, then the curve is vulnerable to attack. It is because of these issues that point-counting on elliptic curves is such a hot topic in elliptic curve cryptography.

**Example:** In the elliptic curve group defined by $y^2 = x^3+9x+17$ over $\mathbb{F}_{23}$, What is the discrete logarithm $k$ of $Q = (4,5)$ to the base $P = (16,5)$?

Ans: One way to find $k$ is to compute multiples of $P$ until $Q$ is found. The first few multiples of $P$ are:

$$\begin{array}{lll} P = (16,5), & 2P = (20,20), & 3P = (14,14), \\ 4P = (19,20), & 5P = (13,10), & 6P = (7,3), \\ 7P = (8,7), & 8P = (12,17), & 9P = (4,5). \end{array}$$

Since $9P = (4,5) = Q$, the discrete logarithm of $Q$ to the base $P$ is $k = 9$.

## 4.2 Elliptic Curve DSA (ECDSA)

This section describes the procedures for generating and verifying signatures using the ECDSA.

- DOMAIN PARAMETER GENERATION

  The domain parameter for ECDSA consist of a suitably chosen elliptic cure $E$ defined over a finite field $\mathbb{F}_p$ of characteristic $p$, and a base point $G \in E_p(a,b)$ with order $n$.

  1. Select a random or pseudo-random integer $x$ such that $1 \leq x \leq n-1$.
  2. Compute $Q = xG$.
  3. A's public key is $Q$; A's private key is $x$.

- ECDSA SIGNATURE GENERATION

  To sign a message $m$, an entity A with domain parameters $(p, E_p(a,b), G, n)$ and associated key pair $(x, Q)$ does the following:

1. Select an integer $k$ such that $1 \leq k \leq n-1$.
2. Compute $kQ = (x_1, y_1)$.
3. Compute $r = x_1 \pmod{n}$. If $r = 0$ then go to step 1.
4. Compute $k^{-1} \pmod{n}$.
5. Compute SHA-1$(m)$ and convert this string to an integer $H(m)$.
6. Compute $s = k^{-1}(H(m) + xr) \pmod{n}$. If $s = 0$, then go to step 1.
7. A's signature for the message $m$ is $(r, s)$.

- ECDSA SIGNATURE VERIFICATION

  To verify A's signature $(r, s)$ on $m$, B obtains an authentic copy of A's domain parameter $(p, E_p(a, b), G, n)$ and associated public key $Q$. B then does the following:

  1. Verify that $r$ and $s$ are integers in the interval $[1, n-1]$.
  2. Compute SHA-1$(m)$ and convert this string to an integer $H(m)$.
  3. Compute $w = s^{-1} \pmod{n}$.
  4. Compute $u_1 = H(m)w \pmod{n}$ and $u_2 = rw \pmod{n}$.
  5. Compute $X = (x_2, y_2) = u_1 G + u_2 Q$.
  6. If $X = \mathcal{O}$, then reject the signature. Otherwise, compute $v = x_2 \pmod{n}$.
  7. Accept the signature if and only if $v = r$.

- PROOF THAT SIGNATURE VERIFICATION WORKS

  If a signature $(r, s)$ on a message $m$ was indeed generated by A, then

$s = k^{-1}(H(m) + xr) \pmod{n}$. Rearranging gives

$$\begin{aligned} kG &= s^{-1}(H(m) + xr)G \pmod{n} \\ &= s^{-1}H(m)G + s^{-1}rxG \pmod{n} \\ &= H(m)wG + rwQ \pmod{n} \\ &= u_1G + u_2Q \pmod{n}. \end{aligned}$$

Thus $u_1G + u_2Q = (u_1 + u_2d)G = kG$, and so $v = r$ as required.

- RATIONALE FOR CHECKS ON $r$ AND $s$ IN SIGNATURE VERIFICATION.

  Step q of signature verification checks that $r$ and $s$ are integers in the interval $[1, n-1]$. These checks can be performed very efficiently, and are prudent measures in light of known attacks on related ElGamal signature schemes which do not perform these checks. The following is a plausible attack on ECDSA if the check $r \neq 0$ (and, more generally, $r \not\equiv 0 \bmod n$) is not performed. Suppose that A is using the elliptic curve $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$, where $b$ is a quadratic residue modulo $p$, and suppose that A uses select a base point $G = (0, \sqrt{b})$ of prime order $n$. (It is plausible that all entities may select a base point with $x$-coordinate in order to minimize the size of domain parameters.) An adversary can now forge A's signature on any message $m$ of its choice by computing $H(m) =$SHA-1$(m)$. It can easily be checked that $(r = 0, s = H(m))$ is a valid signature for $m$.

Comparing DSA and ECDSA. Conceptually, the ECDSA is simply obtain from the DSA by replacing the subgroup of order $q$ of $(\mathbb{Z}/p\mathbb{Z})^{\times}$ generated by $g$ with the subgroup of points on an elliptic curve that are generated by $G$. The only significant difference between ECDSA and DSA is in the generation of $r$. The DSA does this by taking the random element $X = g^k \bmod p$ and reducing it modulo $q$, thus obtaining an integer in the interval $[1, q-1]$. The ECDSA generates $r$ in the interval $[1, n-1]$ by taking the $x$-coordinate of the random point $kG$ and reducing it modulo $n$.

## 4.3 DSA VS. ECDSA

| | DSA | ECDSA |
|---|---|---|
| Key generation | Select $p,\ q,\ x,\ q \mid p-1$,<br>and $1 \le x < q$.<br>Select $h \in [1, p-1]$, compute<br>$g = h^{(p-1)/q} \pmod p$<br>$y = g^x \pmod p$<br>public key : $(p,\ q,\ g,\ y)$<br>private key : $x$ | Select $E_p(a,b),\ x$,<br>and $1 \le x < n$.<br>Select $G \in E_p(a,b)$ with order $n$<br>and compute<br>$Q = dG$<br>public key : $(E_p(a,b), p, G, n, Q)$<br>private key : $x$ |
| Signature generation | Select $k$, $1 \le k < n$.<br>$r = (g^k \bmod p) \pmod q$<br>$s = k^{-1}(H(m) + xr) \pmod q$<br>$(r, s)$ is the signature of $m$. | Select $k$, $1 \le k < q$.<br>$kG = (x_1, y_1), \quad r = x_1 \pmod n$<br>$s = k^{-1}(H(m) + xr) \pmod n$<br>$(r, s)$ is the signature of $m$. |
| Signature verification | $w = s^{-1} \pmod q$<br>$u_1 = H(m)w \pmod q$<br>$u_2 = rw \pmod q$<br><br>$v = (g^{u_1} y^{u_2} \bmod p) \pmod q$<br>$v = r \Rightarrow$ accept the signature. | $w = s^{-1} \pmod n$<br>$u_1 = H(m)w \pmod n$<br>$u_2 = rw \pmod n$<br>$u_1G + u_2Q = (x_2, y_2)$,<br>$v = x_2 \pmod n$<br>$v = r \Rightarrow$ accept the signature. |

## 4.4 An Example of ECDSA

**Example:** Let $p = 114973$; the elliptic curve $E : y^2 = x^3 - 3x + 69424$ and a base point $G = (11570, 42257)$ with order $n = 114467$; select $x = 86109$ then $Q = xG = (6345, 28549)$; and the message $m =$ "worldof" it's hash value $H(m) = 1789679805$, the signature for the message $m$ is $(r, s)$ as following:

- ECDSA SIGNATURE as following:

  1. Select $k = 84430$ such that $1 \leq k \leq n - 1$.
  2. Compute $kG = (11705, 10585), r = 31167 \pmod{114973}$.
  3. Compute $s = k^{-1}(H(m) + xr) = 82722 \pmod{114973}$.

- ECDSA VERIFICATION as following:

  1. Compute $w = s^{-1} = 83035 \pmod{114973}$.
  2. Compute
  $$\begin{aligned} u_1 &= H(m)w = 71001 \pmod{114973} \\ u_2 &= rw = 81909 \pmod{114973} \end{aligned}$$
  3. Compute
  $$\begin{aligned} u_1G &= (66931, 53304) \\ u_2Q &= (88970, 41780), \end{aligned}$$
  $u_1G + u_2Q = (31167, 31627)$ and $v = 31167 \pmod{114973}$.
  4. $$\begin{aligned} v &= 31167 \pmod{114973} \\ r &= 31167 \pmod{114973}. \end{aligned}$$

We obtain $v = r$, that is accept the signature.

# Chapter 5

# A variation of ECDSA

## 5.1 Algorithm

A variant of ECDSA.

- ECDSA DOMAIN PARAMETER GENERATION

  The domain parameter for ECDSA consist of a suitably chosen elliptic cure $E$ defined over a finite field $\mathbb{F}_p$ of characteristic $p$, and a base point $G \in E_p(a, b)$ with order $n$.

  1. Select a random or pseudo-random integer $x$ such that $1 \leq x \leq n - 1$.
  2. Compute $Q = xG$.
  3. A's public key is $Q$; A's private key is $x$.

- ECDSA SIGNATURE GENERATION

  To sign a message $m$, an entity A with domain parameters $(p, E_p(a, b), G, n)$ and associated key pair $(x, Q)$ does the following:

  1. Select two integers $k_1, k_1$ such that $1 \leq k_1,\ k_1 \leq n - 1$.
  2. Compute $k_1G = (x_1, y_1)$ and $k_2G = (x_2, y_2)$.

3. Compute $r_1 = x_1 \pmod n$ and $r_1 = x_2 \pmod n$. If $r_1 = 0$ and $r_1 = 0$ then go to step 1.
4. Compute $k_1^{-1} \pmod n$.
5. Compute SHA-1($m$) and convert this string to an integer $H(m)$.
6. Compute $s = k_1^{-1}(H(m)k_1 + x(r_1 + r_1)) \pmod n$. If $s = 0$, then go to step 1.
7. A's signature for the message $m$ is $(r_1,\ s)$.

- ECDSA SIGNATURE VERIFICATION

To verify A's signature $(r_1, s)$ on $m$, B obtains an authentic copy of A's domain parameter $(p, E_p(a,b), G, n)$ and associated public key $Q$. B then does the following:

1. Verify that $r_1$ and $r_2$ are integers in the interval $[1, n-1]$.
2. Compute SHA-1($m$) and convert this string to an integer $H(m)$.
3. Compute $w = s^{-1} \pmod n$.
4. Compute $u_1 = H(m)wk_2 \pmod n$ and $u_2 = (r_1 + r_2)w \pmod n$.
5. Compute $X = (x_3, y_3) = u_1 G + u_2 Q$.
6. If $X = \mathcal{O}$, then reject the signature. Otherwise, compute $v = x_3 \pmod n$.
7. Accept the signature if and only if $v = r_1$.

- PROOF THAT SIGNATURE VERIFICATION WORKS

If a signature $(r_1, s)$ on a message $m$ was indeed generated by A, then

$s = k_1^{-1}(H(m)k_2 + x(r_1 + r_2)) \pmod{n}$. Rearranging gives

$$\begin{aligned} k_1G &= s^{-1}(H(m)k_2 + x(r_1+r_2))G \pmod{n} \\ &= s^{-1}H(m)k_2G + s^{-1}(r_1+r_2)xG \pmod{n} \\ &= H(m)wk_2G + (r_1+r_2)wQ \pmod{n} \\ &= u_1G + u_2Q \pmod{n}. \end{aligned}$$

Thus $u_1G + u_2Q = (u_1 + u_2d)G = k_1G$, and so $v = r_1$ as required.

## 5.2 EDCSA VS. variation ECDSA

| | ECDSA | variant ECDSA |
|---|---|---|
| Key generation | Select $E_p(a,b),\ x,$ and $1 \le x < n$. Select $G \in E_p(a,b)$ with order $n$ and compute $Q = dG$ public key : $(E_p(a,b), p, G, n, Q)$ private key : $x$ | Select $E_p(a,b),\ x,$ and $1 \le x < n$. Select $G \in E_p(a,b)$ with order $n$ and compute $Q = dG$ public key : $(E_p(a,b), p, G, n, Q)$ private key : $x$ |
| Signature generation | Select $k$, $1 \le k < n$. $kG = (x_1, y_1),\ r = x_1(\text{mod} n)$ $s = k^{-1}(H(m) + xr)(\text{mod} n)$ $(r, s)$ is the signature of $m$. | Select $k_1,\ k_2$, $1 \le k_1,\ k_2 < n$. $k_1G = (x_1, y_1),\ r_1 = x_1(\text{mod} n)$ $k_2G = (x_2, y_2),\ r_2 = x_2(\text{mod} n)$ $s = k_1^{-1}(H(m)k_2 + x(r_1 + r_2))(\text{mod} n)$ $(r_1, s)$ is the signature of $m$. |
| Signature verification | $w = s^{-1}(\text{mod} n)$ $u_1 = H(m)w(\text{mod} n)$ $u_2 = rw(\text{mod} n)$ $u_1G + u_2Q = (x_2, y_2),$ $v = x_2(\text{mod} n)$ $v = r \Rightarrow$ accept the signature. | $w = s^{-1}(\text{mod} n)$ $u_1 = H(m)wk_2(\text{mod} n)$ $u_2 = (r_1 + r_2)w(\text{mod} n)$ $u_1G + u_2Q = (x_3, y_3),$ $v = x_3(\text{mod} n)$ $v = r_1 \Rightarrow$ accept the signature. |

## 5.3 Examples

**Example 1:** Let $p = 114973$; the elliptic curve $E : y^2 = x^3 - 3x + 69424$ and a base point $G = (11570, 42257)$ with order $n = 114467$; select $x = 86109$ then $Q = xG = (6345, 28549)$; and the message $m =$ "worldof" the hash value is $H(m) = 1789679805$, the signature for the message $m$ is $(r_1,\ s)$ as following:

- ECDSA SIGNATURE as following:

  1. Select $k_1 = 32685, k_2 = 43508$ such that $1 \leq k_1,\ k_2 \leq n - 1$.
  2. Compute

$$\begin{aligned} k_1G &= (11705, 10585),\ r_1 = 11705 \pmod{114973} \\ k_2G &= (4060, 59439),\quad r_2 = 4060 \pmod{114973}. \end{aligned}$$

  3. Compute $s = k_1^{-1}(H(m)k_2 + x(r_1 + r_2)) = 31509 \pmod{114973}$.

- ECDSA SIGNATURE as following:

  1. Compute $w = s^{-1} = 71694 \pmod{114973}$.
  2. Compute

$$\begin{aligned} u_1 &= H(m)k_2w = 57445 \pmod{114973} \\ u_2 &= (r_1 + r_2)w = 8752 \pmod{114973}. \end{aligned}$$

  3. Compute

$$\begin{aligned} u_1G &= (83855, 23496) \\ u_2Q &= (37512, 96852), \end{aligned}$$

$$u_1G + u_2Q = (11705, 10585) \text{ and } v = 11705 \pmod{114973}.$$

4.

$$\begin{aligned} v &= 11705 \pmod{114973} \\ r_1 &= 11705 \pmod{114973}. \end{aligned}$$

We obtain $v = r_1$, that is accept the signature.

**Example 2:** Let $p = 150197$; the elliptic curve $E : y^2 = x^3 - 3x + 45624$ and a base point $G = (48640, 94626)$ with order $n = 150033$; select $x = 52414$ then $Q = xG = (15837, 75466)$; and the message $m =$ "ecdsanew from monkey" the hash value is $H(m) = 596493798$, the signature for the message $m$ is $(r_1,\ s)$ as following:

- ECDSA SIGNATURE as following:

  1. Select $k_1 = 18506, k_2 = 56012$ such that $1 \leq k_1,\ k_1 \leq n - 1$.
  2. Compute

$$\begin{aligned} k_1G &= (78866, 50297),\ r_1 = 78866 \pmod{150197} \\ k_2G &= (53820, 3610),\ \ r_2 = 53820 \pmod{150197}. \end{aligned}$$

  3. Compute $s = k_1^{-1}(H(m)k_2 + x(r_1 + r_2)) = 105358 \pmod{150197}$.

- ECDSA VERIFICATION as following:

  1. Compute $w = s^{-1} = 13843 \pmod{150197}$.
  2. Compute

$$\begin{aligned} u_1 &= H(m)k_2w = 50883 \pmod{150197} \\ u_2 &= (r_1 + r_2)w = 68312 \pmod{150197}. \end{aligned}$$

  3. Compute

$$\begin{aligned} u_1G &= (77938, 110891) \\ u_2Q &= (18615, 114143), \end{aligned}$$

$$u_1G + u_2Q = (78866, 50297) \text{ and } v = 78866 \pmod{150197}.$$

4.

$$\begin{aligned} v &= 78866 \pmod{150197} \\ r_1 &= 78866 \pmod{150197}. \end{aligned}$$

We obtain $v = r_1$, that is accept the signature.

## 5.4 Security Analysis

The reason why our scheme is better than original one is following:

1. In the original ECDSA. The secrets $k$ use to sign two or more messages should generated independently of each other. In particular, a different secret $k$ should signed; otherwise, the private key $x$ can be recovered. Note that if a secure random or pseudo-random number generator is used, then the chance of generating a repeated $k$ value is negligible. To see how private keys can be recovered if secrets are repeated, suppose that the same secret $k$ was used to generate ECDSA signatures $(r, s_1)$ and $(r, s_2)$ on two different messages $m_1$ and $m_2$. Then

$$\begin{aligned} s_1 &= k^{-1}(H(m_1) + xr) \pmod{n} \\ s_2 &= k^{-1}(H(m_2) + xr) \pmod{n}, \end{aligned}$$

where $H(m_1) = \text{SHA-1}(m_1)$ and $H(m_2) = \text{SHA-1}(m_2)$. Then

$$\begin{aligned} ks_1 &= H(m_1) + xr \pmod{n} \\ ks_2 &= H(m_2) + xr \pmod{n}. \end{aligned}$$

Subtraction gives $k(s_1 - s_2) = H(m_1) - H(m_2) \pmod{n}$. If $s_1 \neq s_2 \pmod{n}$, which occurs with overwhelming probability, then $k = (s_1 -$

$s_2)^{-1}(H(m_1) - H(m_2)) \pmod{n}$. Thus, an adversary can determine $k$, and then use this to recover $x$.

2. On our scheme, if we use the same secret $k_1,\ k_2$ was used to generate ECDSA signatures $(r_1, s_1)$ and $(r_1, s_2)$ on two different messages $m_1$ and $m_2$. Then

$$\begin{aligned} s_1 &= k_1^{-1}(H(m_1)k_2 + x(r_1 + r_2)) \pmod{n} \\ s_2 &= k_1^{-1}(H(m_2)k_2 + x(r_1 + r_2)) \pmod{n}, \end{aligned}$$

where $H(m_1) = \text{SHA-1}(m_1)$ and $H(m_2) = \text{SHA-1}(m_2)$. Then

$$\begin{aligned} k_1 s_1 &= H(m_1)k_2 + x(r_1 + r_2) \pmod{n} \\ k_1 s_2 &= H(m_2)k_2 + x(r_1 + r_2) \pmod{n}. \end{aligned}$$

Subtraction gives $k_1(s_1 - s_2) = (H(m_1) - H(m_2))k_2 \pmod{n}$. Even if $s_1 \neq s_2 \pmod{n}$, we obtain the relation equation of $k_1(s_1 - s_2) = (H(m_1) - H(m_2))k_2$. We can not determine $k$ by this equation and then use this to recover $x$. Hence, our scheme is more security.

# References

[1] D. Johnson and A. Menezes, "The Elliptic Curve Digital Signature Algorithm(ECDSA)," *Techical Report CORR 99-34,Centre for Applied Cryptograpic Research (CACR), University of Waterloo*, August 1999.

[2] S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme against adaptive choosen message attracks," *SIAM Journal on Computing*, 17 (1997), 281-308.

[3] S. Blake-Wilson and A. Menezes, "Entity authentication and authenticated key transport ptotocols employing asymmetric techiques ," *Proceedings of the 5th International Workshop on Security Protocols, Lecture Notes in Computing Science* **1361**(1984), 137-158.

[4] National Institute of Standards and Technology, *Digital Signature Standard*, FIPS Publication 186, 1994.

[5] National Institute of Standards and Technology, *Entity Authentication using Public Key Cryptofraphy*, FIPS Publication 196, 1997.

[6] M. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," MIT/LCS/TR-212, MIT Laboratory for Computer Science, 1979.

[7] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM* **21**(1978), 120-126.

[8] C. Schnorr, "Efficient signature generation by smart cards" *Journal of Cryptology*, **43** (1991), 161-174.

[9] NIST, DRAFT Special Publication 800-57, "Recommendation on Key Management", http://csrc.nist.gov/CryptoToolkit/kms/guideline-1-Jan03.pdf, 2003.

[10] NIST, FIPS 186-2, "Digital Signature Standard," http://csrc.nist.gov/publiscations/fips/fips186-2/fips186-2-change1.pdf, 2001.

[11] NIST, FIPS PUB 180-2, "Secure Hash Standard," http://csrc.nist.gov/publiscations/fips/fips180-2/fips180-2withchangenotice.pdf, 2002.

[12] NIST, "NIST Brief Comments on Recent Cryptanalytic Attracks on SHA-1," http://csrc.nist.gov/hash_standards_comments.pdf, 2005.

[13] D. Galindo, S. Martin and J.L. Villar, "Evaluating elliptic curve based KEMs in the light of pairings," http://eprint.iacr.org/2004/084.pdf, 2004.

[14] 張草薰, "ElGamal數位簽署的推廣與改良," 東海大學數學研究所碩士論文, 2000.

## Appendix

```
In[1]:= <<NumberTheory‘NumberTheoryFunctions‘
In[2]:= ecadd[p1_,p2_,a_,b_,n_]:=Module[z,m,x3,y3,p3,
        z=0;z1=1;If[p1==”infinity”,”infinity”,p3=p2;z=1,” ”];
        If[z==1,” ”,If[p2==”infinity”,”infinity”,p3=p1;z=1,” ”]];
        If[z==1,” ”, If[p1[[1]]==p2[[1]]&&p1[[2]]==p2[[2]]==0,
        p3=”infinity”,”infinity”;z=1,” ”]];
        If[z==1,” ”,If[p1[[1]]==p2[[1]]&&p1[[2]]≠2[[2]],
        p3=”infinity”,”infinity”;z=1,” ”]];
        If[z==1,” ”, If[p1==p2&&GCD[p1[[2]],n]≠1&&
        GCD[p1[[2]],n]≠n,z=1;
        z1=GCD[p1[[2]],n],” ”]];
        If[z==1,” ”,If[p1==p2,
        m=Mod[(3*p1[[1]]^2+a)*PowerMod[2*p1[[2]],-1,n],n];
        z=1;x3=m^2-p1[[1]]-p2[[1]];
        y3=m*(p1[[1]]-x3)-p1[[2]];p3=Mod[x3,y3,n], ” ”]];
        If[z==1,” ”,If[GCD[p2[[1]]-p1[[1]],n]≠1,z=1;
        z1=GCD[p2[[1]]-p1[[1]],n],” ”]];
        If[z==1,” ”,m=Mod[(p2[[2]]-p1[[2]])*
        PowerMod[p2[[1]]-p1[[1]],-1,n],n];
        x3=m^2-p1[[1]]-p2[[1]];y3=m*(p1[[1]]-x3)-p1[[2]];
        p3=Mod[x3,y3,n]];If[z1==1,p3,”factor=”,z1]];
        ecmus[p1_,m_,a_,b_,n_]:=Module[z,z=p1;
        For[i=1,i¡m&&z[[Length[z]]][[1]]≠”factor=”,i++,
        z=Append[z,ecadd[p1,z[[Length[z]]],a,b,n]]];z]
        ecmlt[p1_,m_,a_,b_,n_]:=Last[ecmus[p1,m,a,b,n]]
```

```
In[3]:= p=114973;
In[4]:= x=11570;
In[5]:= y=42257;
In[6]:= G={x, y};
In[7]:= b=69424;
In[8]:= n=114467;
In[9]:= d=Random[Integer, 1, n - 1]
In[10]:= 86109
In[11]:= Q=Last[multell[G, d, -3, b, p]]
In[12]:= 6345, 28549
In[13]:= H=Hash[worldof]
In[14]:= 1789679805
In[15]:= k=Random[Integer, 1, n - 1]
In[16]:= 32685
In[17]:= q=Random[Integer, 1, n - 1]
In[18]:= 43508
In[19]:= P=Last[multell[G, k, -3, b, p]]
In[20]:= {11705, 10585}
In[21]:= r=Mod[P[[1]], n]
In[22]:= 11705
In[23]:= L=Last[multell[G, q, -3, b, p]]
In[24]:= {4060, 59439}
In[25]:= t=Mod[L[[1]], n]
In[26]:= 4060
In[27]:= s=Mod[PowerMod[k, -1, n]*(H*q + d*(r + t)), n]
In[28]:= 31509
```

```
In[29]:= w=PowerMod[s, -1, n]
In[30]:= 71694
In[31]:= u1=Mod[H*q*w, n]
In[32]:= 57445
In[33]:= u2=Mod[(r + t)*w, n]
In[34]:= 8752
In[35]:= R=Last[multell[G, u1, -3, b, p]]
In[36]:= {83855, 23496}
In[37]:= J=Last[multell[Q, u2, -3, b, p]]
In[38]:= {37512, 96852}
In[39]:= F=addell[R, J, -3, b, p]
In[40]:= {11705, 10585}
In[41]:= v=Mod[F[[1]], n]
In[42]:= {11705, 10585}
In[43]:= v==r
In[44]:= True
```