HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science

Billy Bob Brumley

Efficient Elliptic Curve Algorithms for Compact Digital Signatures

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, November 27, 2006

Supervisor: Prof. Kaisa Nyberg
Instructor: Prof. Kaisa Nyberg

HELSINKI UNIVERSITY
OF TECHNOLOGY

ABSTRACT OF THE
MASTER'S THESIS

Author:	Billy Bob Brumley		
Name of the thesis:	Efficient Elliptic Curve Algorithms for Compact Digital Signatures		
Date:	November 27, 2006	**Number of pages:**	53
Department:	Department of Computer Science and Engineering	**Professorship:**	T-79
Supervisor:	Prof. Kaisa Nyberg		
Instructor:	Prof. Kaisa Nyberg		

Elliptic curves are attractive due to the reduced size of keys and signatures. Improving the verification speed of such signatures has been the subject of much research. This thesis explores compact digital signatures using elliptic curves. A survey of normal basis field multiplication methods is done, as well as memory requirement and performance analysis for software implementation. An easily implementable alternative to τ-adic Joint Sparse Form is presented, as well as an algorithm for generating low-weight joint τ-adic representations of an arbitrary number of integers. A computationally efficient method for carrying out self-certified, identity-based signature verifications is given, as well as a more secure key issuing protocol.

Keywords: Packet Level Authentication, elliptic curve cryptography, identity-based cryptography, self-certified keys, small digital signatures, Koblitz curves, joint sparse form, simultaneous elliptic scalar multiplication

TEKNILLINEN KORKEAKOULU — DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Billy Bob Brumley		
Työn nimi:	Tehokkaita Elliptisten Käyrien Algoritmeja Kompakteille Digitaalisille Allekirjoituksille		
Päivämäärä:	November 27, 2006	**Sivuja:**	53
Osasto:	Tietotekniikan osasto	**Professuuri:**	T-79
Työn valvoja:	Prof. Kaisa Nyberg		
Työn ohjaaja:	Prof. Kaisa Nyberg		

Elliptiset käyrillä toteutettuina digitaalisten allekirjoitusten ja niiden luomiseen tarvittavien avainten pituudet ovat lyhyitä muihin tunnettuihin allekirjoitusmenetelmiin verrattuna. Mutta allekirjoitusten tarkistaminen on hitaampaa ja sen vuoksi mahdollisuuksia tehokkaampaan tarkistamiseen on paljon tutkittu. Tässä diplomityössä tarkastellaan erityisen kompakteja digitaalisia allekirjoituksia elliptisillä käyrillä. Työssä luodaan katsaus normaalikantoja käyttäviin äärellisten kuntien kertolaskualgoritmeihin, sekä niiden ja ohjelmallisten toteutusten tilakompleksisuuteen ja laskennalliseen tehokkuuteen. Työssä on kaksi uutta tulosta. Ensimmäinen on helposti toteutettavissa oleva vaihtoehtoinen algoritmi τ-kantaisen yhteisen harvan esityksen laskemiseen kahdelle kokonaisluvulle. Toinen ja merkittävämpi tulos on algoritmi, jolla voidaan generoida harva yhteisesitys mielivaltaisen monelle kokonaisluvulle.

Näitä tuloksia on sovellettu itsestään varmentuvien allekirjoitusten tarkistamiseen. Itsestään varmentuvissa allekirjoituksissa julkisen avaimen menetelmän tarvitsemat varmenteet on integroitu osaksi allekirjoitusta, mutta toisaalta ne vaativat salaisten avainten muodostamiseen varmenneviranomaisen apua. Työssä on näytetty että erityisesti elliptisillä käyrillä salaisten avainten muodostaminen näitä allekirjoituksia varten voidaan toteuttaa turvallisemmin. Näin työssä on kehitetty ja toteutettu ohjelmallisesti käytännöllinen ja tehokas digitaalinen allekirjoitusmenetelmä, joka täyttää pakettitason autentikoinnin asettamat tehokkuus- ja tilavaatimukset.

Avainsanat: pakettitason autentikointi, elliptisten käyrien salaustekniikka, identiteettiin perustuvat digitaaliset allekirjoitukset, itsestään varmentuvat avaimet, kompaktit digitaaliset allekirjoitukset, Koblitz käyrät, kokonaislukujen harva yhteisesitys, yhtäaikainen elliptinen skalaarilla kertominen

Acknowledgements

This work was supported by the project “Packet Level Authentication” funded by TEKES.

To acknowledge those directly involved with this work, thanks goes to:

- Prof. Kaisa Nyberg for suggestions, comments, and extremely generous support throughout not only this thesis, but my articles and studies as well.
- Kimmo Järvinen for useful comments and suggestions.
- others involved in the PLA project.

In addition, personal thanks goes to:

- my fiancée Hanna Miettinen for everything.
- my family (the one in Finland, too) for support.
- Dr. Laurie Champion (I call her Mom), San Diego State University, for being my role model and mentor.
- my former instructor and adviser Mr. Peter Chase, Sul Ross State University, for guiding me though my BSc studies.
- my former instructor Dr. Raymond Beaulieu, NSA (formerly at SRSU), for sparking my interest in cryptography.

Otaniemi, November 27, 2006

Billy Bob Brumley, `<billy.brumley at hut.fi>`

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Insecurity of the Internet

As Hardin noted [Har68] in "The Tragedy of the Commons," a large, shared resource will inevitably be exploited by its users. This idea is timeless and has been documented as far back as Aristotle (350 BC) [Ari43]:

> For that which is common to the greatest number has the least care bestowed upon it. Every one thinks chiefly of his own, hardly at all of the common interest; and only when he is himself concerned as an individual.

The same is true of the Internet. Attacks such as denial-of-service, distributed denial-of-service, packet spoofing, etc. are widespread. Therefore, new and more efficient protection methods are required. One vision known as *Packet Level Authentication* (PLA) [CC05] is that more protection is needed at the network infrastructure level. The sender should include a digital signature and some addition data in every packet so that other nodes can verify the integrity, timeliness, and uniqueness of packets without previous communication with the sender.

1.2 Packet Level Authentication

As mentioned, PLA seeks to provide protection at the network infrastructure level. More specifically, a digital signature is attached to every packet to allow every hop along the route to verify the authenticity of the packet. This is different from other end-to-end solutions, such as IPSec, where authenticity can only be verified once the packet has reached the final destination. A sample IPv6 PLA packet header is shown in Figure 1.1.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+   <hop-by-hop header>         |   TTP ID                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Seqnum 32 most significant bytes     |
|                          Seqnum 32 least significant bytes    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Certificate (N bits)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Signature   (M bits)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Packet creation time (32 bits)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Payload (up to about 1400 bytes)                   |
|                                                               |
```

Figure 1.1: IPv6 header with PLA fields.

Currently, the Signature field contains an RSA signature on the packet. The Certificate field contains the sender's public key as well as a trusted third party's signature on the public key. Table 1.1 shows that ECC can greatly reduce the size of these fields.

The informal requirements that PLA needs from a digital signature algorithm are summarized in the following criteria.

Criterion 1 *The signature scheme must be secure.* Although this requirement is quite broad, in practice it means that the signature scheme should be built on those that exist in the standards. The threat of attacks such as impersonation, hash function collisions, existential forgery, etc. must be minimal.

Criterion 2 *The signatures and public keys must be compact.* Since a digital signature and certificate (containing the signer's public key and trusted third party's signature) is attached to every packet, it is important to keep the components as small as possible to avoid excess overhead in the packet.

Criterion 3 *The signatures must be able to be generated and verified quickly.* Since the authenticity of every packet needs to be verified, the selected signature scheme must perform in a manner as to not cause excess latency in the network.

Criterion 4 *The solution must be viable.* The goal is to obtain a practical solution that is useable in the real world. Therefore, the solution must be scalable and practical for a software and/or hardware implementation.

1.3 Digital Signatures and Level of Security

Digital signatures provide a means for authentication of many types of digital data. There are many different signature schemes in existence and still more being developed from year to year. Digital signatures are based on some type of difficult mathematical problem. Therefore, it is important to have different types of signature schemes to provide alternatives in the event that any of these given problems becomes easier to solve. There are three categories outlined in [IEE99]; these are briefly stated below.

Integer Factorization (IF). Given a composite number $n = pq$, p, q sufficiently large primes, it is difficult to find p, q given only n. Example: RSA [RSA78].

Discrete Log (DLP). Given an integer a relatively prime to n and g a primitive root of n, it is difficult to find b such that $a = g^b \pmod{n}$. Examples: Digital Signature Algorithm (DSA) [Kra93], Schnorr [Sch91], Nyberg-Rueppel [NR93].

Elliptic Curve Discrete Log (ECDLP). Given a generator G and the point $P = kG$, it is difficult to find the scalar k. Examples: those from DLP, but using elliptic curve groups.

The difficulty of solving these problems varies. Table 1.1 from [BW05] shows an equivalent level of security (in bits) against attacks using current methods. Note that the elliptic curve schemes have much lower key sizes than other public key primitives; this is the main advantage of using elliptic curves, and the reason that elliptic curve cryptography (ECC) is the focus of this thesis.

Strength	ECC	DSA/RSA
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

Table 1.1: Comparable key sizes (in bits).

As computing power increases, so does the level of security. Table 1.2 lists the key sizes that are currently recommended by NIST [BBB+05].

Years	Strength	DSA	RSA	EC
now-2010	80	1024/160	1024	160
2011-2030	112	2048/224	2048	224
2030+	128	3072/256	3072	256

Table 1.2: Recommended minimum key sizes (in bits).

1.4 Contributions

The solution herein for PLA combines the use of Koblitz curves, elliptic curve digital signatures, self-certified keys, and simultaneous scalar multiplication. The Nyberg-Rueppel signature scheme is used, which is present in many common standards. A secure self-certified key issuing protocol is also used. This satisfies Criterion 1. Criteria 2 and 3 are a tradeoff. While RSA signature verifications can be much faster than those using elliptic curves [Sco06], Table 1.2 shows that RSA signatures are much larger than signature schemes that use elliptic curves. This difference will be even greater in the future. Therefore, the use of elliptic curves is imperative. The size and computational efficiency is further improved by the use of self-certified keys, Koblitz curves, and simultaneous scalar multiplication. These methods were selected and designed with software and hardware implementation in mind, and hence Criterion 4 is satisfied.

PLA is just one example of a security problem in which digital signatures, if implemented efficiently, can provide a solution. PLA provides the motivation; however, the results and contributions are applicable to many areas of elliptic curve cryptography. The focus is on software implementation, but many of the presented algorithms will perform well in hardware, too. The tangible contributions of this thesis are listed below.

Low-Weight Joint τ-adic Representations. An algorithm for generating low-weight, signed-bit τ-adic representations of an arbitrary number of integers is presented. This combines and extends the ideas of τ-adic Joint Sparse Form [CLSQ03] and the Binary Signed Digit representation [RK04].

Fast Signature Verifications Using Self-Certified Keys. A method for combining self-certified key extraction and signature verifications is presented. This is accomplished using Shamir's Trick [ElG85] combined with the above algorithm to reduce the joint weight.

Secure Self-Certified Key Issuing Protocol. A modification to the blind key issuing protocol in [AdM04] is presented, in which the use of elliptic curves allows for the elimination of the proof of knowledge step. Eliminating this step reduces the complexity of the protocol, as one roundtrip communication is saved.

Software Implementation. The solution herein has been implemented in the C programming language. Although there has been some optimization for speed, this software should mostly be considered as proof-of-concept, as the cryptographic settings have been selected with hardware performance in mind. In the PLA project, most of this software will eventually be replaced by hardware. The PLA software (including the crypto implementation described) can be found at `http://www.tcs.hut.fi/Software/PLA/` .

1.5 Outline

Binary fields are covered in Chapter 2. A survey of normal basis field multiplication methods is done, as well as efficiency analysis for software implementation.

Elliptic curves are covered in Chapter 3. The basics of elliptic curve arithmetic are described, such as elliptic scalar multiplication. An efficient alternative to

τ-adic Joint Sparse Form is presented, as well as an algorithm for low-weight joint τ-adic representations of an arbitrary number of integers.

Digital signatures are covered in Chapter 4. A review of the Nyberg-Rueppel signature scheme is given, as well as a similar self-certified identity-based signature scheme. A computationally efficient method for carrying out these signature verifications is presented, using the contributions from Chapter 3. A more secure self-certified key issuing protocol is also provided.

Chapter 2

Binary Fields

"What a wealth, what a grandeur of thought may spring from what slight beginnings." (referring to groups) *Henry F. Baker*, [Caj91]

Given a positive integer m known as the *degree*, the binary finite field $\mathbb{F}_{2^m}$ is made up of the 2^m possible strings of bits of length m. For software and hardware implementations, binary field arithmetic can be very efficient as it typically involves lots of bitwise operations..

Addition. To compute the sum c of two elements $a, b \in \mathbb{F}_{2^m}$, a, b are added bitwise modulo 2. Therefore, $c = a \oplus b$, where $\oplus$ denotes the bitwise XOR operation.

The definition of other more complex operations (e.g., multiplication, inversion, squaring, etc.) depends on the representation of the binary field elements.

2.1 Binary Field Element Representations

A *representation* of a binary field element determines how a bit string is to be interpreted. There are two popular methods that are described below.

2.1.1 Polynomial Basis

A *polynomial basis* representation uses each bit as a coefficient of a polynomial having degree of at most $m-1$. Formally, the bit string $\langle a_{m-1} \dots a_1 a_0 \rangle$ is interpreted as the polynomial

$$a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + a_1x + a_0 \ .$$

A *field polynomial* is an irreducible polynomial of degree m. When the product of two field elements is computed, the result is then reduced modulo the field polynomial.

2.1.2 Normal Basis

Given a root β of a field polynomial, a *normal basis* representation for the elements of the binary field $\mathbb{F}_{2^m}$ makes use of the set of m linearly independent elements

$$\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-2}}, \beta^{2^{m-1}}\} .$$

A set is said to be linearly independent if no subset adds to zero. Formally, the bit string $\langle a_0, a_1, \ldots, a_{m-1} \rangle$ is interpreted as the element

$$a_0\beta + a_1\beta^2 + a_2\beta^{2^2} + \cdots + a_{m-2}\beta^{2^{m-1}} + a_{m-1}\beta^{2^{m-1}} .$$

Multiplication using normal bases is not as straight-forward as the polynomial basis case. However, a *Gaussian* normal basis (GNB) exists when m is not divisible by eight. Multiplication using a GNB is much more efficient than arbitrary normal bases. Therefore, only GNBs will be considered here.

Each GNB has a type[1] T associated with it which describes the complexity of multiplication; the lower the value of T, the easier multiplication is. A GNB with $T = 1, 2$ is known as a type I or type II *optimal* normal basis (ONB), respectively. Multiplication is most efficient when an ONB is used. Only values of m for which T is even will be considered here.

Squaring. When using a normal basis, one advantage is that squaring is virtually free. Given an element $a = \langle a_0, a_1, \ldots, a_{m-2}, a_{m-1} \rangle \in \mathbb{F}_{2^m}$, the computation a^2 involves only a right rotation[2] of the bits, denoted $\gg$. More formally,

$$a^2 = \langle a_{m-1}, a_0, a_1, \ldots, a_{m-3}, a_{m-2} \rangle .$$

Note that this is a bijection. Given an integer k, the notation $a \gg k$ (k right rotations) is the computation of a^{2^k}. It follows that the computation $\sqrt{a}$ involves only a left rotation of the bits, denoted $\ll$:

$$\sqrt{a} = \langle a_1, a_2, \ldots, a_{m-2}, a_{m-1}, a_0 \rangle .$$

[1]More details on the type of a GNB can be found in [IEE99]

[2]This is also known as a barrel shift or circular shift operation (CSO).

2.2 Normal Basis Multiplication

The main disadvantage of normal basis representation over polynomial basis representation is the speed of multiplication. Normal basis representation has traditionally been shunned in software implementations due to lack of efficient multiplication algorithms. Many standards and references omit normal basis representation entirely [SEC00, HMV04].

Most software packages fail to include support for normal basis representation as well. Crypto++ [Dai06], LiDIA [LG06], and MIRACL [Sco06] are common cryptographic libraries that include elliptic curve cryptography. However, none of them support normal basis representation, only polynomial basis. The implementation in [Ros99] is the one notable exception, which includes support for ONBs, but not for other GNBs.

The cryptographic software implementation for the PLA project is intended as a "proof-of-concept". Most, if not all of the cryptographic software will eventually be replaced by hardware components. Therefore, the software has been written to model a system that will be computationally efficient in hardware, but not necessarily in software. For this reason, only GNB representations are considered. However, given these restraints the software should still perform as well as possible, meaning efficient methods for GNB arithmetic are still important.

2.2.1 Bit-Level GNB Multiplication

Only one method for multiplication when using a GNB in presented in [IEE99]. The sequence $F(1), F(2), \ldots, F(p-1)$ defines the multiplication rule for a GNB. Given integers $p = mT + 1$ and u of order $T \pmod p$, $F(k)$ is generated by

$$F(2^i u^j \mod p) = i \text{ , where } 0 \leq i \leq m-1 \text{ , } 0 \leq j < T \text{ .} \tag{2.1}$$

It should only be generated once, offline at initialization (not at each iteration of a field multiplication) as it is fixed for a given value of m. An example $F(k)$ sequence for $m = 7$, $T = 4$, $p = 29$ is presented in Table 2.1.

For the product $c = ab$, the coefficient c_0 is computed as

$$c_0 = \sum_{k=1}^{p-2} a_{F(k+1)} b_{F(k)} \text{ ,} \tag{2.2}$$

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$F(k)$	0	1	5	2	1	6	5	3	3	2	4	0	4	6
k	15	16	17	18	19	20	21	22	23	24	25	26	27	28
$F(k)$	6	4	0	4	2	3	3	5	6	1	2	5	1	0

Table 2.1: Example $F(k)$ sequence for $m = 7$, $T = 4$, $p = 29$.

The other coefficients are calculated by left rotations on the formula for c_0. The result is Algorithm 1.

Algorithm 1: Bit-level GNB multiplication.

```
Input: Field elements $a, b \in \mathbb{F}_{2^m}$
Output: The product $c = ab \in \mathbb{F}_{2^m}$
$c \leftarrow 0$
for $i \leftarrow 0$ to $m - 1$ do
    for $k \leftarrow 1$ to $p - 2$ do  $c_i \leftarrow c_i \oplus (a_{F(k+1)} b_{F(k)})$
    $a \leftarrow a \ll 1, b \leftarrow b \ll 1$
end
return $c$
```

For a practical software implementation, the normal basis multiplication method in Algorithm 1 is virtually unusable. Accessing arbitrary bits is not a trivial task in software (as opposed to hardware). A typical software implementation of field elements for elliptic curve cryptography uses the following structure [DHH+04] presented as Figure 2.1, where W is the computer word size and $A_{\lfloor m/W \rfloor}$ is padded on the right with zeros.

$A_0 : a_0 a_1 \ldots a_{W-1}$	$A_1 : a_W a_{W+1} \ldots a_{2W-1}$	$\ldots$	$A_{\lfloor m/W \rfloor} : \ldots a_{m-1}$

Figure 2.1: Data structure for a binary field element.

Therefore, accessing an arbitrary bit of a field element requires division, modular reduction, shifting, and some bitwise operations. At each iteration of the inner loop in Algorithm 1, up to three bits need to be accessed, as well as two table lookups. For one field multiplication, $O(m^2 T)$ executions of the inner loop are needed, so the algorithm does not perform well in software.

2.2.2 Vector-Level GNB Multiplication

Recently, however, much progress has been made in the development of normal basis multiplication algorithms that are more software-oriented [NY01, RMH01, RMH03, FD04, DHH+04, RM06]. These algorithms operate at the vector-level instead of the bit-level. In practice, this means that operations are done on the W-bit words of a field element instead of accessing arbitrary bits. Bitwise operations on computer words are generally very efficient. Algorithm 2 (modified from [RMH03]) demonstrates vector-level GNB multiplication. Bitwise AND is denoted by $\odot$.

Algorithm 2: Vector-level GNB multiplication.

```
**Input**: Field elements $A, B \in \mathbb{F}_{2^m}$
**Output**: The product $C = AB \in \mathbb{F}_{2^m}$
$S_A \leftarrow A, S_B \leftarrow B, C \leftarrow 0$
**for** $k \leftarrow 1$ **to** $p - 2$ **do**
    $S_A \lll F(k)$
    $C \leftarrow C \oplus (S_A \odot S_B)$
    $S_B \lll F(k)$
**end**
**return** $C$
```

As the values S_A, S_B are not being modified (they are just rotations of A, B respectively), these rotations can be precomputed. This idea was first introduced in [NY01]. While their approach was for general normal bases and extended to ONBs, the idea has led to multiple enhancements for field multiplication for GNBs [DHH+04, RM06].

2.2.3 The Ning-Yin Method

Using general normal bases, multiplication is normally carried out using an $m \times m$ matrix M known as the *multiplication matrix*. In the GNB case, M can be easily computed using the sequence $F(k)$ from Equation 2.1 as

$$\textbf{for}\ \ i \leftarrow 1 \textbf{ to } p-2 \textbf{ do } M_{F(p-i),F(i+1)} \leftarrow M_{F(p-i),F(i+1)} + 1 \pmod 2\ . \tag{2.3}$$

Since $F(k)$ is fixed for a given m and generated offline, it follows that M is also fixed and should be generated offline. An example of M for $\mathbb{F}_{2^7}$ is shown in Figure 2.2.

$$\begin{array}{|ccccccc|}
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 1
\end{array}$$

Figure 2.2: Multiplication matrix M for $\mathbb{F}_{2^7}$.

The approach in the Ning-Yin method [NY01] is to store[3] m rotations of A, B. Let $A_i = A^{2^{m-i}}$ (sequential left rotations of A) and consider the example of naively storing $m = 163$ rotations of A when $W = 32$ presented in Figure 2.3.

	TA			
$A^{2^{163}} = A_0 =$	$a_0 \dots a_{31}$	$a_{32} \dots a_{63}$	$\dots$	$a_{160} \dots a_{162}$
$A^{2^{162}} = A_1 =$	$a_1 \dots a_{32}$	$a_{33} \dots a_{64}$	$\dots$	$a_{161} \dots a_0$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$A^{2^{131}} = A_{32} =$	$a_{32} \dots a_{63}$	$a_{64} \dots a_{95}$	$\dots$	$a_{29} \dots a_{31}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$A^{2^1} = A_{162} =$	$a_{162} \dots a_{30}$	$a_{31} \dots a_{62}$	$\dots$	$a_{159} \dots a_{161}$

Figure 2.3: Storing $m = 163$ rotations of A.

This requires $m \times \lceil m/W \rceil$ computer words. However, there are many redundant words being stored in memory; for example, word 1 of A_0 = word 0 of A_{32}. The storage requirements can be reduced by eliminating these redundant words. More specifically, let TA_i be word 0 of A_i (TA is column 0 of Figure 2.3). The rotation A_i can then be accessed via the words

$$A_i = TA_i \ , \ TA_{i+W} \ , \ TA_{i+2W} \ , \ \dots \ , \ TA_{i+\lfloor m/W \rfloor W}$$

[3]This does not necessarily mean m rotations need be performed; if the rotation operation is considered non-trivial, then at most only W rotations are needed.

For example, in Figure 2.3

$$A_0 = TA_0 \ , TA_{32} \ , TA_{64} \ , TA_{92} \ , TA_{128} \ , TA_{160}$$

(clearing the extra bits at the end). This method of storage only requires m words. The modular reductions can be eliminated by doubling the size of TA in a wrap-around fashion. Algorithm 3 from [NY01] demonstrates this method.

Algorithm 3: Ning-Yin vector-level normal basis multiplication.

```
Input: field elements $A, B$
Output: field element $C = AB$
Precompute arrays $A_i, B_i$
$C \leftarrow 0$
for $i \leftarrow 0$ to $m - 1$ do
    $S \leftarrow 0$                    /* go L-R ($j$), T-B ($i$) through M */
    for $j \leftarrow 0$ to $m - 1$ do
        if $M_{i,j} = 1$ then $S \leftarrow S \oplus B_j$
    end
    $C \leftarrow C \oplus (A_i \odot S)$
end
return $C$
```

2.2.4 The Improved Ning-Yin Method

It was proved in [RM06] that the multiplication matrix M has at most T non-zero entries in each row. An improvement to the Ning-Yin method was suggested which makes use of an $m \times T$ matrix R which holds the indices of the ones in M. An example of R for $\mathbb{F}_{2^7}$ is shown in Figure 2.4. Since M is fixed and generated offline, so is R.

This solution, presented as Algorithm 4, is very attractive due to the drastic reduction in the amount of shifting (at the cost of storing the tables TA, TB). Note that there is an inner loop present bound by the type T.

2.2.5 Normal Basis Multiplication Costs

Field multiplication costs measured by the number of bitwise operations are presented in Table 2.2. Two algorithms are compared; the bit-level Algorithm 1 and the vector-level Algorithm 4. The field sizes of $m = 163, 233$ are analyzed, as well as the computer word sizes $W = 32, 64$.

1	0	0	0
0	2	5	6
1	3	4	5
2	5	0	0
2	6	0	0
1	2	3	6
1	4	5	6

Figure 2.4: Matrix R for $\mathbb{F}_{2^7}$.

Algorithm 4: Improved Ning-Yin vector-level GNB multiplication.

```
**Input**: field elements $A, B$
**Output**: field element $C = AB$
Precompute arrays $A_i, B_i$
$C \leftarrow A \odot B_1$
**for** $i \leftarrow 1$ **to** $m - 1$ **do** $C \leftarrow C \oplus (A_i \odot (B_{R(i,1)} \oplus B_{R(i,2)} \oplus \ldots \oplus B_{R(i,T)}))$
**return** $C$
```

As demonstrated, bit-level and vector-level GNB multiplication algorithms are fundamentally different. As shown in Figure 2.1, the lookup of arbitrary bits in software often involves some more complex calculations. These costs are not considered in Table 2.2. The value for the Lookups column includes both table lookups as well as (in the vector-level case) precomputed rotation lookups (each computer word counting as one lookup).

There are many points to consider when viewing Table 2.2. Higher values for the computer word size are always more efficient. Increasing from $m = 163$ to $m = 233$ has a very small impact on the number of bitwise operations. This is due mainly to the multiplication complexity type (T) difference. As shown in Tables 1.1 and 1.2, this result is significant because as the security requirements increase in the future, this means there will be very little field multiplication efficiency loss when increasing the field size from $m = 163$ to $m = 233$.

Method	Words (d)	Shifts	ANDs	XORs	Lookups
Bit-level (BL)	-	$2m$	Tm^2	Tm^2	Tm^2
Vector-level (VL)	$\lceil m/W \rceil$	$2m$	dm	$dT(m+1)$	$m(T(d+1)+d)$
BL, $m = 163, T = 4$	-	326	106276	106276	106276
VL, $W = 32$	6	326	978	3936	5542
VL, $W = 64$	3	326	489	1968	3097
BL, $m = 233, T = 2$	-	466	108578	108578	108578
VL, $W = 32$	8	466	1864	3744	5592
VL, $W = 64$	4	466	932	1872	3262

Table 2.2: Estimated field multiplication costs.

2.3 Normal Basis Inversion

Using the fact that squaring is essentially free, the inverse of a field element can be calculated using exponentiation as

$$a^{-1} = a^{-2}a = a^{-2}a^{2^m} = a^{2^m-2} \tag{2.4}$$

While a general exponentiation algorithm can be used, a specialized algorithm was presented in [IT88] for exponentiation to the power of $2^m - 2$. The version from [IEE99] is outlined below in Algorithm 5.

Algorithm 5: Normal basis inversion by exponentiation.

```
Input: Field element $a \in \mathbb{F}_{2^m}$.
Output: The inverse $a^{-1}$.
Let $b_i$ denote coefficient $i$ in the binary expansion of $m-1$ and $b_j$ the most
significant bit.
$e \leftarrow a$ , $k \leftarrow 1$
for $i \leftarrow j-1$ to 0 do
    $c \leftarrow e \gg k$                /* $c \leftarrow e^{2^k}$, $k$ right-rotations of $e$ */
    $e \leftarrow ce$ , $k \leftarrow 2k$
    if $b_i = 1$ then $e \leftarrow e^2 a$ , $k \leftarrow k+1$
end
return $e^2$
```

Since Algorithm 5 is bound by the fixed value of m, the cost of inversion using normal bases is fixed; these costs are presented in Table 2.3 for a few different values of m. The costs are exact, as the number of field multiplications is dependent only on the value of m. As inversion in $\mathbb{F}_p$ is estimated at a cost of 80 field multiplications [HMV04], inversion in $\mathbb{F}_{2^m}$ is much more efficient.

m	T	Mults
113	2	8
131	2	8
163	4	9
233	2	10

Table 2.3: Normal basis inversion costs for different binary field sizes ($\mathbb{F}_{2^m}$).

Chapter 3

Elliptic Curves

"It is possible to write endlessly on elliptic curves.
(This is not a threat.)" *Serge Lang*, [Lan78]

Elliptic curves for cryptographic use [Mil86, Kob87] are defined by their *Weierstrass equation*. These curves are over a finite field $\mathbb{F}_q$, where $q = p$ (a prime finite field) or $q = 2^m$ (a binary finite field) [Kob94, IEE99]. An elliptic curve over $\mathbb{F}_p$ is the set of points $P = (x, y)$ satisfying the Weierstrass equation

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

where $x, y \in \mathbb{F}_p$. Similarly, an elliptic curve over $\mathbb{F}_{2^m}$ is the set of points satisfying the Weierstrass equation

$$y^2 + xy = x^3 + ax^2 + b \ , \tag{3.2}$$

where $x, y \in \mathbb{F}_{2^m}$. Another point called the *point at infinity*, denoted by ∞, is the identity element ($\infty + P = P$). The *order* of E is the number of points on E including ∞, denoted $\#E(\mathbb{F}_q)$. Elliptic curves for cryptographic use have large prime order r.

While it is possible to generate elliptic curves with large prime order, in practice standardized curves are often used. For example, NIST published [NIS99] many elliptic curves for cryptographic use that have been adopted by various cryptographic standards [ANS98, SEC00]. Implementations have been well documented [BHLM01, HHM00].

As the implementation here is targeted towards software and hardware, elliptic curves over $\mathbb{F}_{2^m}$ will be the focus, as binary field arithmetic is much more efficient in software and hardware than prime field arithmetic. However, elliptic curves over prime fields are generally easier to visualize geometrically. Hence, many of the small examples will use elliptic curves over prime fields.

The following notation will be used for operation costs: S = field squaring, M = field multiplication, I = field inversion, D = point doubling, A = point addition.

3.1 Point Addition and Doubling on $E(\mathbb{F}_{2^m})$

For adding two distinct points $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ on an elliptic curve over $\mathbb{F}_{2^m}$, the following equation is used.

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \text{ , where} \\ \lambda &= \frac{y_1 + y_2}{x_1 + x_2} \end{aligned} \tag{3.3}$$

This computation requires[1] $1\mathsf{S} + 2\mathsf{M} + 1\mathsf{I}$. To subtract the point $P = (x, y)$ the point $-P = (x, x + y)$ is added.

To double a point $2(x_1, y_1) = (x_3, y_3)$ on an elliptic curve over $\mathbb{F}_{2^m}$, the following equation is used.

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + a \\ y_3 &= x_1^2 + x_3(\lambda + 1) \text{ , where} \\ \lambda &= x_1 + \frac{y_1}{x_1} \end{aligned} \tag{3.4}$$

This requires $2\mathsf{S} + 2\mathsf{M} + 1\mathsf{I}$.

3.2 Projective Coordinates

As inversion is generally more expensive than multiplication, projective coordinates are often used to trade a certain number of field multiplications for inversion. While [IEE99] advocates the use of Jacobian projective coordinates for all $E(\mathbb{F}_q)$, *López-Dahab* (LD) projective coordinates [LD99] actually prove to be a little more efficient for $E(\mathbb{F}_{2^m})$.

[1]Division is inversion followed by multiplication.

3.2.1 LD Coordinates

Using LD coordinates, the point $(X : Y : Z)$, where $Z \neq 0$, corresponds to the affine point $(X/Z, Y/Z^2)$. The point at infinity $\infty = (1 : 0 : 0)$. The projective elliptic curve equation is given by

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \ . \tag{3.5}$$

Point Addition, Mixed Coordinates. Given $P = (X_1 : Y_1 : 1)$ and $Q = (X_2 : Y_2 : Z_2)$, where $P \neq \pm Q$, $P + Q = (X_3 : Y_3 : Z_3)$ is calculated [ADMRK02] as

$$\begin{aligned} &U = Z_2^2Y_1 + Y_2 \ , \ S = Z_2X_1 + X_2 \ , \ T = Z_2S \ , \ Z_3 = T^2 \ , \\ &V = Z_3X_1 \ , \ C = X_1 + Y_1 \ , \ X_3 = U^2 + T(U + S^2 + aT) \ , \\ &Y_3 = (V + X_3)(TU + Z_3) + Z_3^2C \ . \end{aligned} \tag{3.6}$$

This requires $8\mathsf{M} + 5\mathsf{S}$ and 8 field additions[2]. Note that P is in affine coordinates while Q and the resulting point are in LD coordinates.

Point Doubling. Given $P = (X_1 : Y_1 : Z_1)$, $2P = (X_3 : Y_3 : Z_3)$ is calculated as

$$\begin{aligned} &S = X_1^2 \ , \ T = Z_1^2 \ , \ Z_3 = ST \ , \ T = bT^2 \ , \\ &X_3 = S^2 + T \ , \ Y_3 = (Y_1^2 + aZ_3 + T)X_3 + TZ_3 \ . \end{aligned} \tag{3.7}$$

This requires $4\mathsf{M} + 5\mathsf{S}$ and 4 field additions. Note that P and the resulting point are both in LD coordinates.

3.3 Elliptic Scalar Multiplication

Elliptic scalar multiplication, the elliptic curve analogue of modular exponentiation [Knu98], replaces all the multiplicative group operations of multiplication and squaring with the analogous elliptic curve operations of point addition and point doubling. Given an ℓ-bit scalar $k = \langle k_{\ell-1} \ldots k_0 \rangle$, $k_i \in \{1, 0\}$ (binary expansion) and a point P, k multiples of P are computed using

$$kP = \sum_{i=0}^{\ell-1} k_i 2^i P \ . \tag{3.8}$$

[2]Assuming $a \in \{0, 1\}$, which is the case for standardized curves [NIS99].

The *Double-and-Add Method* is outlined in Algorithm 6. The average number of non-zero digits in ℓ-bit k is $\ell/2$, so it is executed at the average cost of

$$\frac{\ell}{2}\mathsf{A} + \ell\mathsf{D}\ . \tag{3.9}$$

Algorithm 6: Right-to-left elliptic scalar multiplication.

```
Input: integer $k$, point $P \in E(\mathbb{F}_q)$
Output: $kP$
$Q \leftarrow \infty$
while $k > 0$ do
    if $k$ is odd then $Q \leftarrow Q + P$                    /* k & 1 */
    $k \leftarrow \lfloor k/2 \rfloor$                          /* right shift by one */
    $P \leftarrow 2P$                               /* using a point doubling method */
end
return $Q$
```

Using Mixed Coordinates

In Algorithm 6, the point P accumulates doubles of P and has a value of 2^iP at each iteration i. Depending on the binary digit of k_i, Q gets the point 2^iP added to it. Projective coordinates should not be used in this case, as both P and Q would be in projective coordinates.

However, when moving left-to-right, the point Q accumulates doubles at each iteration, then depending on the digit k_i has the point P is added to Q. Having P in affine coordinates and Q in LD coordinates is then appropriate and projective coordinates can be used. Hence, algorithms that move left-to-right are generally preferred over right-to-left. However, right-to-left methods are usually easier to understand and express logically, so many of the examples here and in academic literature present right-to-left methods for brevity.

3.3.1 NAF, Addition-Subtraction Method

In multiplicative groups, an unsigned binary representation is used for exponentiation, since inversion is much more expensive than multiplication. However, for elliptic scalar multiplication a binary signed-digit representation known as *Non-Adjacent Form* (NAF) [Gor98] of the ℓ-bit scalar $k = \langle k_{\ell-1} \dots k_0 \rangle$, $k_i \in \{1, -1, 0\}$

is commonly used. For groups over elliptic curves, the analogous operation of point subtraction has roughly the same cost as point addition [MO90].

As with the unsigned binary representation, each integer has a unique NAF. Of any two adjacent digits, at least one must be zero. Given an ℓ-bit integer in NAF, the average density is $\ell/3$, which is minimal among all signed binary representations.

Deriving the NAF of an integer is very similar to computing the normal, unsigned binary representation of an integer. For the unsigned representation, the integer is repeatedly divided by 2, outputting the remainder at each step (either 0 or 1). NAF is generated by repeatedly dividing k by 2, choosing a remainder such that the quotient is divisible by 2.

The *Addition-Subtraction Method* presented as Algorithm 7 from [Sol00] is the only method present in many standards [IEE99, ANS98]. It performs the scalar multiplication and NAF calculation simultaneously. This is done to avoid the extra temporary storage of the NAF representation. Since there are on average $\ell/3$ non-zero digits using NAF, Algorithm 7 is executed at the cost of

$$\frac{\ell}{3}\mathsf{A} + \ell\mathsf{D} \ . \tag{3.10}$$

Algorithm 7: Right-to-left elliptic scalar multiplication using NAF.

```
**Input**: integer $k$, point $P \in E(\mathbb{F}_q)$
**Output**: $kP$
$Q \leftarrow \infty$
while $k > 0$ do
    if $k$ is odd then
        $u \leftarrow 2 - (k \mod 4)$          /* get last 2 binary digits */
        $k \leftarrow k - u$
        if $u = 1$ then $Q \leftarrow Q + P$
        if $u = -1$ then $Q \leftarrow Q - P$
    end
    $k \leftarrow \lfloor k/2 \rfloor$                      /* right shift by one */
    $P \leftarrow 2P$                  /* using a point doubling method */
end
return $Q$
```

An example of Algorithm 7 is presented in Figure 3.1. When using an unsigned binary representation,

$$1262 = 2^{10} + 2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 \ .$$

Using NAF,

$$1262 = 2^{10} + 2^{8} - 2^{4} - 2^{1} \ .$$

2^i	k_{bin}	bin	Q_{bin}	k_{NAF}	NAF	Q_{NAF}
0	1262	0	∞	1262	0	∞
1	631	1	$\infty + 2P = 2P$	631 - $\bar{1}$ = 632	$\bar{1}$	$\infty + -2P = -2P$
2	315	1	$2P + 4P = 6P$	316	0	
3	157	1	$6P + 8P = 14P$	158	0	
4	78	0		79 - $\bar{1}$ = 80	$\bar{1}$	$-2P + -16P = -18P$
5	39	1	$14P + 32P = 46P$	40	0	
6	19	1	$46P + 64P = 110P$	20	0	
7	9	1	$110P + 128P = 238P$	10	0	
8	4	0		5 - 1 = 4	1	$-18P + 256P = 238P$
9	2	0		2	0	
10	1	1	$238P + 1024P = 1262P$	1 - 1 = 0	1	$238P + 1024P = 1262P$
			7 Additions			4 Additions

Figure 3.1: Computation of $1262P$ using binary and NAF.

3.3.2 Combing

Elliptic scalar multiplication by the *Combing Method* [Pro03, HMV04] can be an efficient alternative to Algorithms 6 and 7. The ℓ-bit scalar k is broken up into w pieces of length $d = \lceil \ell / w \rceil$. All possible values of

$$a_{w-1}2^{(w-1)d}P + \ldots + a_2 2^{2d}P + a_1 2^{d}P + a_0 P \ , \text{ where } a \in 0, 1, -1$$

are then precomputed. The d columns are then processed in a left-to-right manner as outlined in Algorithm 8. An example is provided in Figure 3.2.

Due to the precomputation requirements (mainly the $d(w-1)$ point doublings), combing is usually only done for fixed values of P, such as the generator G. The precomputation can then be done offline and persisted. In this case (offline precomputation), Algorithm 8 is executed at the cost of

$$\left(1 - \frac{2^w}{3^w}\right) d\mathsf{A} + d\mathsf{D} \qquad (3.11)$$

Algorithm 8: Left-to-right elliptic scalar multiplication by combing.

```
Input: $\ell$-bit integer $k$ in NAF, point $P \in E(\mathbb{F}_q)$
Output: $kP$
Precompute $a_{w-1}2^{(w-1)d}P + \ldots + a_2 2^{2d}P + a_1 2^d P + a_0 P \;\forall\; a_i \in \{0, -1, 1\}$
$Q \leftarrow \infty$
for $i \leftarrow d-1$ to $0$ do
    $Q \leftarrow 2Q$
    $Q \leftarrow Q + (k_{dw-i}2^{d(w-1)}P + k_{d(w-1)-i}2^{(w-2)d}P + \cdots + k_{d-i}P)$
end
return $Q$
```

$$\begin{aligned}
\mathrm{NAF}(2524) =& \overbrace{1010}^{2^8}\overbrace{00\bar{1}0}^{2^4}\overbrace{0\bar{1}00}^{2^0} \\
=& 1010 \cdot (2^8) + \\
& 00\bar{1}0 \cdot (2^4) + \\
& 0\bar{1}00 \cdot (2^0) \\
=& 2^3 \cdot 2^8 + 2^2 \cdot (-2^0) + 2 \cdot (2^8 - 2^4) \\
=& 2(2(2 \cdot 2^8 - 2^0) + 2^8 - 2^4)
\end{aligned}$$

	computing: **2524P** $\ell = 12, d = 4, w = 3$ precompute $a_2 2^8 P + a_1 2^4 P + a_0 P \;\forall\; a_i \in \{0, -1, 1\}$ (10 points) $Q \leftarrow \infty$
$i = 3$	$Q \leftarrow 2(\infty) = \infty$ $Q \leftarrow \infty + 2^8 P = 256P$
$i = 2$	$Q \leftarrow 2(256P) = 512P$ $Q \leftarrow 512P - P = 511P$
$i = 1$	$Q \leftarrow 2(511P) = 1022P$ $Q \leftarrow 1022P + (2^8 P - 2^4 P) = 1262P$
$i = 0$	$Q \leftarrow 2(1262P) = 2524P$ $Q \leftarrow 2524P + \infty =$ **2524P**

Figure 3.2: Combing example.

3.4 Simultaneous Elliptic Scalar Multiplication

In many signature verification primitives, the main operation often involves a calculation similar to

- $g_1^k g_2^l$ for multiplicative groups.
- $kP + lQ$ for groups over elliptic curves.

The straight-forward method is to calculate each term separately, then combine the result. Using Algorithm 7, this requires on average

$$\left(\frac{2\ell}{3}+1\right)\mathsf{A}+2\ell\mathsf{D}\ . \tag{3.12}$$

However, calculations of the above form have specialized methods known as *simultaneous* elliptic scalar multiplication. Using a modification of *Shamir's Trick* [ElG85] to process $kP + lQ$ in parallel can reduce the number of operations needed. This method was first documented in [Str64] and is sometimes referred to as *Straus's Algorithm*.

The idea is that the values for the individual terms are not needed, only their sum. It works by precomputing all possible values of a column in a joint representation, then moves left-to-right, performing elliptic curve operations at each step. See Figure 3.3 for a small example.

Algorithm 9 modified from [HMV04] illustrates this method. Note that if k_i, l_i are both zero, no point addition takes place (the point at infinity is added). Improvements known as *Window* methods [HMV04] (looking at more than one digit of the scalars at each iteration) can also be used, but as the window size increases along with the use of NAF the amount of precomputation required causes substantial diminishing returns.

When using NAF on the pair of integers (k, l) the probability of a non-zero column is $1 - (2^2/3^2) = 5/9$, giving Algorithm 9 an average cost of

$$\left(\frac{5\ell}{9}+2\right)\mathsf{A}+\ell\mathsf{D} \tag{3.13}$$

including precomputation and assuming point negation is free. This is a substantial improvement over processing the elliptic scalar multiplications separately.

	computing: $\mathbf{13P + 7Q}$ precomp: $(P+Q), (P-Q)$ NAF(13) $= 10\bar{1}01$ $(2^4 - 2^2 + 2^0 = 13)$ NAF(7) $= 0100\bar{1}$ $(2^3 - 2^0 = 7)$ $R \leftarrow \infty$
$i = 4$	$R \leftarrow 2R = \infty$ $R \leftarrow \infty + P = P$
$i = 3$	$R \leftarrow 2P$ $R \leftarrow 2P + Q$
$i = 2$	$R \leftarrow 2(2P+Q) = 4P + 2Q$ $R \leftarrow 4P + 2Q - P = 3P + 2Q$
$i = 1$	$R \leftarrow 2(3P+2Q) = 6P + 4Q$ $R \leftarrow 6P + 4Q + \infty = 6P + 4Q$
$i = 0$	$R \leftarrow 2(6P+4Q) = 12P + 8Q$ $R \leftarrow 12P + 8Q + (P-Q) = \mathbf{13P + 7Q}$

Figure 3.3: Small example of Shamir's Trick, computing $13P + 7Q$ $(\bar{1} = -1)$.

Algorithm 9: Left-to-right simultaneous elliptic scalar multiplication.

```
Input: $\ell$-bit integers $k, l$, points $P, Q \in E(\mathbb{F}_q)$
Output: $kP + lQ$
Precompute $xP + yQ \ \forall \ x, y \in \{0, -1, 1\}$
Compute a signed binary-digit representation of $k, l$
$R \leftarrow \infty$
for $i \leftarrow \ell - 1$ to $0$ do
    $R \leftarrow 2R$
    $R \leftarrow R + (k_i P + l_i Q)$
end
return $R$
```

3.4.1 Joint Sparse Form

Another representation was developed in [Sol01] called *Joint Sparse Form* (JSF), which is a generalization of NAF for a pair of integers. JSF has minimal *joint weight* (JW) among all signed binary digit representations for a pair of integers, yielding an average of $\ell/2$ non-zero columns. The example from [Sol01] is presented in Table 3.1, showing that NAF has a higher JW than JSF. The second row is the original JSF, while the third row is a left-to-right alternative to JSF from [RK04].

Form	Representation	JW	Length
NAF	$53 = 010\bar{1}0101$		
	$102 = 10\bar{1}010\bar{1}0$	8	8
JSF	$53 = 0100\bar{1}0\bar{1}\bar{1}$		
	$102 = 100\bar{1}\bar{1}0\bar{1}0$	6	8
l2r BSD	$53 = 0100\bar{1}\bar{1}01$		
	$102 = 011010\bar{1}0$	6	7

Table 3.1: Binary signed digit joint representations.

Therefore, when using JSF, Algorithm 9 is slightly more efficient than NAF, requiring

$$\left(\frac{\ell}{2} + 2\right) \mathsf{A} + \ell \mathsf{D} \ . \tag{3.14}$$

3.4.2 Generalization of Joint Sparse Form

Thusfar, simultaneous elliptic scalar multiplication via Shamir's Trick has only been considered for two points. However, more generally it can be used for computations involving n scalars $(k_0, \ldots, k_{n-1})$ and elliptic curve points $(P_0, \ldots, P_{n-1})$:

$$\sum_{i=0}^{n-1} k_i P_i \ . \tag{3.15}$$

As JSF is defined for a pair of integers, a generalization of JSF to n terms can be used to reduce the joint weight. The joint weight of n integers in a joint representation is defined as the number of columns with at least one non-zero entry.

Solinas [Sol01] suggested a generalization of JSF as future work, with a remark questioning the practicality due to increased precomputation requirements. Such a generalization was presented independently in [Pro03, GHP04]. The need for a left-to-right method of generating a low-weight representation of an arbitrary number of integers was also noted, so as to work in-line with Shamir's Trick and not require separate storage of the joint representation, which was presented in [RK04].

3.5 Koblitz Curves

Koblitz curves [Kob92] are anomalous binary curves of the form

$$E_a : y^2 + xy = x^3 + ax^2 + 1 \text{ where } a \in \{0, 1\} \ . \tag{3.16}$$

Since $E_a(\mathbb{F}_2)$ is a subgroup of $E_a(\mathbb{F}_{2^m})$, the cofactor $f = \#E_a(\mathbb{F}_2)$ divides $\#E_a(\mathbb{F}_{2^m})$. If m is prime, then $r = \#E_a(\mathbb{F}_{2^m})/f$ can also be prime. Koblitz curves with large prime r are suitable for cryptographic use. Those points of order r are said to be in the *main subgroup*. A few examples of such Koblitz curves are listed in Table 3.2.

m	a	$\#E_a(\mathbb{F}_{2^m})$
131	0	$2^2 \cdot 680564733841876926932320129493409985129$
163	1	$2 \cdot 5846006549323611672814741753598448348329118574063$
233	0	$2^2 \cdot 3450873173395281893717377931138512760570940988862252126328087024741343$

Table 3.2: Examples of Koblitz curves.

The *Frobenius map* $\tau : E_a(\mathbb{F}_{2^m}) \to E_a(\mathbb{F}_{2^m})$ is a mapping such that $(x, y) \mapsto (x^2, y^2)$. From Equation 3.16 it follows that if (x, y) is on the curve then so is (x^2, y^2). Squaring an element of $\mathbb{F}_{2^m}$ is a very cheap operation. It can be shown from Equation 3.3 that for all $(x, y) \in E_a$

$$\begin{aligned} (x^4, y^4) + 2(x, y) &= \mu(x^2, y^2) \ , \text{ where } \mu = (-1)^{1-a} \ , \text{ or,} \\ (\tau^2 + 2)P &= \mu\tau P \ , \text{ from where} \\ \tau^2 + 2 &= \mu\tau. \end{aligned} \tag{3.17}$$

Using elliptic scalar multiplication along with the Frobenius map, it is possible to compute a multiple of a point by using complex multiplication by an element of the ring $\mathbb{Z}[\tau]$. More specifically, instead of representing integer k as distinct powers of 2, k is represented as the sum of distinct powers of τ, called a τ-adic expansion of k. For example, $9 = \tau^5 - \tau^3 + 1$ when $a = 1$ as shown in Figure 3.4.

Analogous to NAF, τ-adic NAF is generated by repeatedly dividing k by τ, choosing a remainder such that the quotient is divisible by τ. Given Equation 3.17, every element of the ring $\mathbb{Z}[\tau]$ can be written in canonical form $c_0 + c_1\tau$. It was proved in [Sol00] that $c_0 + c_1\tau$ is divisible by τ if and only if c_0 is even, and divisible by τ^2 if and only if $c_0 \equiv 2c_1 \pmod 4$. Analogous to Equation 3.8, multiples of the point P are then computed as

$$kP = \sum_{i=0}^{\ell_\tau - 1} k_i \tau^i P \tag{3.18}$$

where ℓ_τ is the length of the τ-adic expansion of k. The τ-adic analogue of Algorithm 7 is shown in Algorithm 10. The τ-adic NAF is generated inline as illustrated in Figure 3.4. Unfortunately, it was shown in [MS92] that while τ-adic NAF has density of $\ell/3$, the length is twice as long as the binary signed NAF. Since no point doublings are required, it is executed at the cost of

$$\frac{2\ell}{3}\mathsf{A} \ . \tag{3.19}$$

Algorithm 10: Right-to-left elliptic scalar multiplication using τ-adic NAF.

```
Input: integer $k$, point $P \in E_a(\mathbb{F}_{2^m})$
Output: $kP$
$Q \leftarrow \infty$ , $c_0 \leftarrow k$ , $c_1 \leftarrow 0$                    /* $k = c_0 + c_1\tau$ */
while $c_0 \neq 0$ or $c_1 \neq 0$ do
    if $c_0$ is odd then
        $u \leftarrow 2 - (c_0 - 2c_1 \mod 4)$
        $c_0 \leftarrow c_0 - u$
        if $u = 1$ then $Q \leftarrow Q + P$
        if $u = -1$ then $Q \leftarrow Q - P$
    end
    $P \leftarrow \tau P$                              /* square the coordinates */
    $c_0 \leftarrow c_1 + \mu c_0/2$ , $c_1 \leftarrow -c_0/2$
end
return $Q$
```

3.5.1 Reduced τ-adic NAF

Fortunately, a method for avoiding the excess length of the τ-adic NAF exists [MS92]. Given any point $P \in E_a(\mathbb{F}_{2^m})$, it follows that

$$\begin{aligned} P &= (x, y) = (x^{2^m}, y^{2^m}) = \tau^m P \\ \infty &= (\tau^m - 1)P \ . \end{aligned}$$

$k_{\tau-NAF}$	c_0	c_1
1	$9-1=8$, $0+8/2=4$	$-8/2=-4$
0	$-4+4/2=-2$	$-4/2=-2$
0	$-2+-2/2=-3$	$--2/2=1$
-1	$-3--1=-2$, $1+-2/2=0$	$--3/2=1$
0	$1+0=1$	0
1	$1-1=0$	0

Figure 3.4: Generating τ-adic NAF for $k=9$.

The elements $\gamma,\rho \in \mathbb{Z}[\tau]$ such that $\gamma \equiv \rho \pmod{\tau^m - 1}$ are said to be *equivalent* with respect to P as γ multiples of P can also be obtained using the element ρ since

$$\gamma P = \rho P + \kappa(\tau^m - 1)P = \rho P + \kappa\infty = \rho P \ .$$

Therefore, scalars can be reduced by $(\tau^m - 1)$ to reduce the length of the τ-adic representation.

When Koblitz curves are used for cryptographic purposes, only points in the main subgroup (those of order r) are considered. Since $(\tau - 1)$ divides $(\tau^m - 1)$, Solinas [Sol00] showed that for points in the main subgroup, reduction by $(\tau^m - 1)/(\tau - 1)$ is possible, leading to slightly shorter τ-adic representations. It was also shown that the average weight of the resulting reduced τ-adic NAF is $\ell/3$ (the same as binary signed NAF). Therefore, when performing modular reduction, Algorithm 10 is executed at the cost of

$$\frac{\ell}{3}\mathsf{A} \ , \tag{3.20}$$

which is a large improvement over the binary signed NAF case.

3.5.2 Koblitz Curves and Combing

It was noted that when combing using Algorithm 8, the amount of point doublings needed in the precomputation stage made it unsuitable for elliptic scalar multiplications of an arbitrary point P. However, when using Koblitz curves this precomputation requirement is significantly reduced as the point doublings are replaced by applications of τ. Therefore, combing can be a very efficient method when used with Koblitz curves.

3.5.3 Joint Sparse Form, τ-adic

In [CLSQ03], a τ-adic analogue of JSF was presented which, like JSF, moves right-to-left. Table 3.3 continues the example provided therein. The first entry demonstrates that simply using τ-adic NAF of both scalars is not optimal with regards to the JW. The second entry is the τ-adic JSF representation with lower JW. The third entry is related to Algorithm 11 and will be explained in Section 3.5.4.

Form	Representation	JW	Length
τ-NAF [Sol00]	$\langle\bar{1}0\bar{1}0\bar{1}0101\rangle$ $\langle0\bar{1}0\bar{1}00010\rangle$	8	9
τ-JSF [CLSQ03]	$\langle\bar{1}00\bar{1}1001\bar{1}\rangle$ $\langle0\bar{1}0\bar{1}00010\rangle$	6	9
Algorithm 11	$\langle0\bar{1}10\bar{1}001\bar{1}\rangle$ $\langle0\bar{1}0\bar{1}00010\rangle$	6	8

Table 3.3: τ-adic joint representations.

3.5.4 An Efficient Alternative to τ-adic Joint Sparse Form

As shown, Shamir's Trick moves left to right through the expansions of (k_0, k_1). Therefore, just as with the binary signed JSF, a method of generating a τ-adic JSF that moves left-to-right would work in-line with Shamir's Trick and also be more memory efficient, as separate storage for the joint representation would not longer be needed. This improved algorithm is presented as Algorithm 11. The strategy is to create more zero columns given the fact that $\tau^2 + 1 = \tau - 1$. The first **for** loop generates the digits and the bottom **while** loop performs the elliptic curve arithmetic. This algorithm works when $\mu = 1$, but only small modifications are needed when $\mu = -1$.

The third entry in Table 3.3 is the left-to-right representation, which happens to demonstrate how moving left-to-right can also decrease the length of the representation by one if substitutions can be made in the first column (meaning up to one point addition and one application of τ can be saved).

Algorithm 11: left to right τ-adic simultaneous elliptic scalar multiplication

```
Input: $\ell$-bit integers $k_0, k_1$ in $\tau$-NAF, points $P, Q \in E(\mathbb{F}_{2^m})$
Output: $k_0P + k_1Q$
Precompute $xP + yQ \ \forall \ x, y \in \{0, -1, 1\}$
$R \leftarrow \infty$, $i \leftarrow \ell - 1$
while $i \geq 0$ do
    $j \leftarrow 1$                                /* number of columns to process */
    for $n \leftarrow 0$ to $1$ do
        if $i > 1$ and $k_{n,i} + k_{n,i-2} = \pm 2$ and $k_{1-n,i} = 0$ then
            $k_{n,i-1} \leftarrow k_{n,i}$                    /* replace the bits */
            $k_{n,i-2} \leftarrow -k_{n,i}$
            $k_{n,i} \leftarrow 0$                        /* zero out the column */
            $j \leftarrow 2$                        /* two columns can be processed */
        end
    end
    while $j > 0$ do
        $R \leftarrow \tau R$                     /* square the coordinates of $R$ */
        $R \leftarrow R + (k_{0,i}P + k_{1,i}Q)$
        $i \leftarrow i - 1$ , $j \leftarrow j - 1$
    end
end
return $R$
```

3.5.5 Generalizing τ-adic Signed-Bit Joint Representations

When using τ-adic representations for computations similar to Equation 3.15, a method for generating a low-weight signed-bit τ-adic joint representation would be useful. In the binary signed digit case, the algorithm in [RK04] works on the fact that for all $n > 0$

$$2^n - 1 = \sum_{i=0}^{n-1} 2^i \ . \tag{3.21}$$

As demonstrated previously with NAF and τ-adic NAF, τ-adic analogues are usually constructed by finding the equivalent operation when working with powers of τ.

Unfortunately, the τ-adic analogue of Equation 3.21 is not immediately apparent as opposed to the NAF case. To produce a τ-adic analogue, for all $n > 1$ the task is to find a solution to one of

$$\tau^n \pm 1 = \sum_{i=0}^{n-1} x_i \tau^i \ , \text{ where } x_i \in \{1, -1, 0\}. \tag{3.22}$$

It turns out that one has four different cases to consider with respect to the value of $n \pmod 4$. Given Equation 3.17 and assuming $\mu = 1$, the following solutions are obtained to Equation 3.22 for the initial cases of $n = 2, 3, 4,$ or 5, leading to Theorem 3.5.1.

$$\begin{aligned}
&\tau^2 + 1 = \tau - 2 + 1 = \tau - 1\\
&\tau^3 + 1 = \tau(\tau^2) + 1 = \tau^2 - 2\tau + 1 = \tau - 2 - 2\tau + 1 = -\tau - 1\\
&\tau^4 - 1 = (\tau^2 + 1)(\tau^2 - 1) = (\tau - 1)(\tau^2 - 1) = \tau^3 - \tau^2 - \tau + 1\\
&\tau^5 - 1 = \tau(\tau^4) - 1 = \tau^4 - \tau^3 - \tau^2 + 2\tau - 1 = \tau^4 - \tau^3 + \tau + 1
\end{aligned}$$

Theorem 3.5.1. *Given an arbitrary $n > 1$ and assuming $\mu = 1$, one of $\tau^n \pm 1$ can be expressed by the equation below depending on the value $k = n \pmod 4$*

$$\tau^n + S_k = \sum_{I_k}^{n-1} \tau^i - \sum_{J_k}^{n-2} \tau^i \text{ , where} \tag{3.23}$$

k	S_k	I_k	J_k
0	-1	$i = 0 \mid i \equiv 0,3 \pmod 4$	$i = 0 \mid i \equiv 1,2 \pmod 4$
1	-1	$i = 0 \mid i \equiv 0,1 \pmod 4$	$i = 3 \mid i \equiv 2,3 \pmod 4$
2	1	$i = 0 \mid i \equiv 1,2 \pmod 4$	$i = 0 \mid i \equiv 0,3 \pmod 4$
3	1	$i = 3 \mid i \equiv 2,3 \pmod 4$	$i = 0 \mid i \equiv 0,1 \pmod 4$

Proof. Only the case $n \equiv 0 \pmod 4$ is proved here, as the other cases are similar. The proof will be by induction. The base case of $n = 4$ holds as

$$\tau^4 - 1 = \tau^3 - \tau^2 - \tau + 1 \text{ .}$$

Assume that, for an arbitrary $k > 1$ satisfying $k \equiv 0 \pmod 4$, the following formula holds:

$$\tau^k = \left(\sum_{i=0 \mid i \equiv 0,3 \pmod 4}^{k-1} \tau^i \right) - \left(\sum_{i=0 \mid i \equiv 1,2 \pmod 4}^{k-2} \tau^i \right) + 1$$

Then the inductive step of $k + 4$ yields

$$\begin{aligned}
&\left(\sum_{i=0|i\equiv 0,3 \pmod 4}^{k+3} \tau^i\right) - \left(\sum_{i=0|i\equiv 1,2 \pmod 4}^{k+2} \tau^i\right) + 1\\
=&\left(\sum_{i=0|i\equiv 0,3 \pmod 4}^{k-1} \tau^i\right) - \left(\sum_{i=0|i\equiv 1,2 \pmod 4}^{k-2} \tau^i\right) + 1 + \tau^{k+3} - \tau^{k+2} - \tau^{k+1} + \tau^k\\
=&\tau^k + \tau^{k+3} - \tau^{k+2} - \tau^{k+1} + \tau^k\\
=&\tau^k(\tau^3 - \tau^2 - \tau + 2) = \tau^k(\tau^4) = \tau^{k+4} \ .
\end{aligned}$$

Therefore, the inductive step also holds. Since the base case and the inductive step are both true, the theorem holds. □

The $\mu = -1$ Case

Given Equation 3.17 and assuming $\mu = -1$, the following solutions to Equation 3.22 are obtained for the initial cases of $n = 2, 3, 4,$ or 5, leading to Theorem 3.5.2.

$$\begin{aligned}
&\tau^2 + 1 = -\tau - 2 + 1 = -\tau - 1\\
&\tau^3 - 1 = \tau(\tau^2) - 1 = -\tau^2 - 2\tau - 1 = -(-\tau - 2) - 2\tau - 1 = -\tau + 1\\
&\tau^4 - 1 = (\tau^2 + 1)(\tau^2 - 1) = (-\tau - 1)(\tau^2 - 1) = -\tau^3 - \tau^2 + \tau + 1\\
&\tau^5 + 1 = \tau(\tau^4) + 1 = -\tau^4 - \tau^3 + \tau^2 + 2\tau + 1 = -\tau^4 - \tau^3 + \tau - 1
\end{aligned}$$

Theorem 3.5.2. *Given an arbitrary $n > 1$ and assuming $\mu = -1$, one of $\tau^n \pm 1$ can be expressed by the equation below depending on the value $k = n \pmod 4$*

$$\tau^n + S_k = \sum_{I_k}^{n-3} \tau^i - \sum_{J_k}^{n-1} \tau^i + T_k \text{ , } \textit{where} \tag{3.24}$$

k	S_k	T_k	I_k	J_k
0	-1	0	$i = 0 \mid i \equiv 0,1 \pmod 4$	$i = 2 \mid i \equiv 2,3 \pmod 4$
1	1	$-\tau^2$	$i = 1 \mid i \equiv 1,2 \pmod 4$	$i = 0 \mid i \equiv 0,3 \pmod 4$
2	1	0	$i = 2 \mid i \equiv 2,3 \pmod 4$	$i = 0 \mid i \equiv 0,1 \pmod 4$
3	-1	τ^2	$i = 0 \mid i \equiv 0,3 \pmod 4$	$i = 1 \mid i \equiv 1,2 \pmod 4$

The proofs are similar to those of Theorem 3.5.1.

3.5.6 Joint τ-adic Representations of n Integers

Now that solutions are known for Equation 3.22, a generalized left-to-right algorithm for generating a low-weight signed-bit τ-adic joint representation of n integers is presented as Algorithm 12. Note that Algorithm 9 is the explicit case of $n = 2$. This algorithm assumes $\mu = 1$. For the $\mu = -1$ case, modifications to steps 1c, 2b, and 2c should be made corresponding to Theorem 3.5.2.

A comparison of the probabilities of a non-zero column given n different integers when using Algorithm 12 is presented in Table 3.4. These values correspond to the number of point additions needed (not including precomputation). Values in the Alg. 12 column are estimates from [Pro03]; values for $n = 1, 2, 3$ have been verified by simulation, while simulation results for $n > 3$ are forthcoming.

n	τ-adic	τ-NAF	Alg. 12
1	.5	.3333	.3333
2	.75	.5555	.5
3	.875	.7037	.5897
4	.9375	.8025	.6425
5	.9688	.8683	.6727
6	.9844	.9122	.6999

Table 3.4: Probabilities of a non-zero column given n terms.

Algorithm 12: Generating a τ-adic joint representation of n integers.

Input: n ℓ-bit integers k_n in τ-NAF expansion

Output: Low-weight signed-bit τ-adic joint representation of k_n

1. Scan the ℓ columns $\times$ n rows from left to right. For each non-zero entry in the column, determine if that row is reducible.

   (a) Count the number of consecutive zeros (denoted C) rightward from the non-zero entry (x). Examine at most n bits. Since all k_n are in τ-NAF, note that $C \geq 1$. If $C \geq n$, then the row **is not** reducible.

   (b) Check the C columns of the n rows rightward from x. If there already exists at least one all-zero column in the next C columns, then the current non-zero column **is not** reducible.

   (c) Determine reducibility as follows:

      i. $C + 1 \equiv 2, 3 \pmod 4$. If the bits from x to the next non-zero entry (x') are of the form $x0...0x$ (same sign), the row **is** reducible.

      ii. $C + 1 \equiv 0, 1 \pmod 4$. If the bits from x to x' are of the form $x0...0\overline{x}$ (opposite sign), the row **is** reducible.

2. If all rows with non-zero entries are determined to be reducible, then perform the replacement in each of the rows to zero-out the column as follows.

   (a) Replace x with 0.

   (b) Replace the bit to the right of x with x.

   (c) For the next C bits (meaning up to and including x'), repeat the pattern $\overline{xx}xx\overline{xx}xx...\overline{xx}xx$ (two x of opposite sign, two x of same sign, two x of opposite sign $\ldots$).

   (d) If C is even, replace the bit two to the left of x' with 0 (e.g., $x\overline{xx} \rightarrow 0\overline{xx}$).

3. If replacements were made, continue scanning again from step 1 after skipping $C + 1$ columns (start scanning again from bit x'). All of the column between are not reducible due to the consecutive bits inserted above. If replacements were not made, check the next column (rows in the current column were not reducible or already zero).

Chapter 4

Digital Signatures

"The Theory of Groups is a branch of mathematics in which one does something to something and then compares the result with the result obtained from doing the same thing to something else, or something else to the same thing." *J. R. Newman*, [New56]

Elliptic curves over finite fields can be used to construct secure cryptosystems that have many advantages over those that use multiplicative groups. These advantages include, but are not limited to, small key and signature sizes. Table 4.1 from [IEE99] outlines elliptic curve analogues of multiplicative groups.

	Multiplicative Groups	Elliptic Curve Groups
Setting	$\mathbb{F}_q$	curve E over $\mathbb{F}_q$
Basic operation	multiplication in $\mathbb{F}_q$	addition of points
Main operation	exponentiation	scalar multiplication
Base element	generator g	base point G
Base element order	prime r	prime r
Private key	s (integer $\mod r$)	s (integer $\mod r$)
Public key	w (element of $\mathbb{F}_q$)	W (point on E)

Table 4.1: Multiplicative and elliptic curve group analogues.

4.1 The Nyberg-Rueppel Signature Scheme Using Elliptic Curves

The Nyberg-Rueppel signature scheme is a variation of the ElGamal scheme [ElG85]. It is one of the few schemes present in many popular standards [IEE99]. An elliptic curve analogue of the Nyberg-Rueppel signature scheme is shown below.

Setup. Elliptic curve E is chosen with base point generator G of prime order r where $r \mid \#E$.

Keygen. Alice generates a private key s and public key W by computing

$$\begin{aligned} s &\in_R \mathbb{Z}_r^* \\ W &= sG \end{aligned} \tag{4.1}$$

This requires one elliptic scalar multiplication involving a fixed point G.

Sign. To generate a signature (c, d) on a message m, Alice calculates

$$\begin{aligned} u &\in_R \mathbb{Z}_r^* \\ c &= [uG]_x + \mathrm{H}(m) \pmod r \\ d &= u - sc \pmod r \end{aligned} \tag{4.2}$$

where $[P]_x$ denotes the x-coordinate of the point P converted to an integer and H is a collision-resistant hash function.

Verify. To verify the signature (c, d) on the message m, Bob checks that

$$\mathrm{H}(m) = c - [dG + cW]_x \pmod r \tag{4.3}$$

Correctness. These computations are consistent:

$$\begin{aligned} dG + cW = dG + csG = (d + cs)G = (u - sc + sc)G = uG \\ c - [uG]_x = [uG]_x + \mathrm{H}(m) - [uG]_x = \mathrm{H}(m) \end{aligned}$$

Alice's public key W is made up of the x-coordinate x_W and the y-coordinate y_W, both of size q. To further reduce this size requirement W can be compressed[1], re-

[1]There are either zero or two solutions to the elliptic curve equation for the y-coordinate when given an x-coordinate. The compression bit determines which solution to use.

quiring only $q + 1$ bits. Both parts of the signature (c, d) are of size r. Signature generation requires one elliptic scalar multiplication while signature verification requires one elliptic scalar multiplication involving a fixed point G and one involving an arbitrary point W for a total of two elliptic scalar multiplications.

4.2 Self-Certified Keys and Signatures

Self-certified keys [Gir91] provide a good alternative to traditional certificate-based PKI. Instead of verifying the certificate and signature separately, the signer's public key is extracted from the trusted third party's signature on the signer's identity and then used to verify the signature. This reduces the computational requirements. Instead of two elliptic scalar multiplications for each of two signatures, only one is needed for the public key extraction and two for the signature verification. The space requirements are also reduced, as an explicit signature on a user's public key is no longer needed.

SC signatures have the following drawback. It is impossible for a third party to verify an extracted public key; if a signature fails to verify, it is unknown where the failure lies. The public key and/or the signature is incorrect.

The concept of a trusted third party can be fairly vague when discussing self-certified keys. To better define the notion of trust, Girault introduced three distinct *trust levels*.

Trust Level 1. TTP knows the user's private key and can therefore impersonate the user in an undetectable manner.

Trust Level 2. TTP does not know the user's private key, but can still impersonate the user in an undetectable manner.

Trust Level 3. TTP does not know the user's private key, but can impersonate the user. However, such impersonation is detectable.

In this case, detectable means that if TTP tries to impersonate a user, the user can prove it; for example, providing two different signatures from TTP on the same identity.

Trust Level 1 is inadequate for many reasons, one being that it usually requires a secure key escrow. Reaching Trust Level 3 is generally the goal; consider the following scenario. An Internet Service Provider (ISP, the user's TTP) charges based on bandwidth usage. Each packet is digitally signed by the user, providing assurance that the ISP is billing in an honest manner. If the ISP can impersonate the user in an undetectable manner, the ISP can generate false traffic from the user to increase the charges. Trust Levels 1 and 2 are therefore inadequate. This is just one example of why Trust Level 3 is desirable.

A self-certified identity based (SCID) signature scheme based on the Nyberg-Rueppel signature scheme was presented in [AdM04]. The paper uses multiplicative group notation, although it is mentioned that the scheme was designed for elliptic curve implementation. The paper also focuses on provable security and as a result, exponentiation of the message hash takes place. A version using groups over elliptic curves is outlined below. Message hash exponentiation is not used, as the focus is not proveable security.

Setup. Elliptic curve E is chosen with base point generator G of prime order r where $r \mid \#E$. The Trusted Third Party (TTP) uses Equation 4.1 to generate a domain private key s_D and domain public key W_D. TTP then publishes W_D.

Keygen. To generate a private key on user Alice's identity ID_A, TTP calculates

$$\begin{aligned} u &\in_R \mathbb{Z}_r^* \\ ([uG]_x, b_A) &= \textsc{compress}(uG) \\ r_A &= [uG]_x + \textsc{h}(ID_A) \pmod{r} \\ s_A &= u - s_D r_A \pmod{r} \end{aligned} \tag{4.4}$$

and escrows the private key s_A to Alice securely and values (r_A, b_A) publicly. COMPRESS is the point compression function, yielding the x-coordinate of uG and the compression bit b_A.

Sign. To generate the signature (c, d) on the message m, Alice uses Equation 4.2.

Verify. After extracting Alice's public key W_A using `Extract`, Bob verifies the signature (c, d) using Equation 4.3.

Extract. To extract Alice's public key W_A on identity ID_A given public values (r_A, b_A), Bob calculates

$$W_A = \textsc{decompress}(r_A - \textsc{h}(ID_A), b_A) - r_A W_D \tag{4.5}$$

where DECOMPRESS is the point decompression function given an x-coordinate and compression bit b. This requires one elliptic scalar multiplication and one point addition.

Correctness. The extracted public key is correct ($W_A = s_A G$):

$$\begin{aligned} W_A &= \textsc{decompress}(r_A - \textsc{h}(ID_A), b_A) - r_A W_D \\ &= uG - r_A s_D G = (u - r_A s_D)G \\ &= (s_A + r_A s_D - r_A s_D)G = s_A G \end{aligned}$$

Note that (r_A, s_A) is simply a Nyberg-Rueppel signature by TTP on the message $m = ID_A$; s_A acts as Alice's private key while r_A will be used by third parties to reconstruct Alice's public key $W_A = s_A G$ as shown in `Extract`. The key issuing protocol `Keygen` only reaches Trust Level 1.

4.3 Improving the Performance of SCID Signatures

Using the above SCID scheme, the extraction of the signer's public key is accomplished using Equation 4.5 and the signature is then verified using Equation 4.3. However, these two equations can be combined to produce

$$\textsc{h}(m) = c - [dG + c(\textsc{decompress}(r_A - \textsc{h}(ID_A), b) - r_A W_D)]_x \pmod{r}$$

Performing the point decompression (producing the point uG) and distributing c yields the calculation

$$dG + c(uG) - cr_A W_D \tag{4.6}$$

Hence, the public key extraction and signature verification process can be rewritten as the sum of three distinct elliptic scalar multiplications. This can be computed most efficiently using three-term simultaneous scalar multiplication.

Using Algorithm 12 (the case of $n = 3$ and $\mu = -1$) with Algorithm 9 for three points, the explicit solution is presented as Algorithm 13. This combination greatly reduces the number of elliptic curve operations needed.

Table 4.2 shows a comparison of the number of required elliptic curve operations (including precomputation) for a few common standardized curves [NIS99] when processing Equation 4.6 using simultaneous and separate elliptic scalar multiplications. The Koblitz curve estimates are for Algorithm 13, while the prime and binary

Algorithm 13: Three-Term τ-adic simultaneous scalar multiplication.

```
Input: $\ell$-bit integers $a, b, c$ in $\tau$-NAF, points $P, Q, R \in E(\mathbb{F}_{2^m})$
Output: $aP + bQ + cR$
Precompute $xP + yQ + zR \;\forall\; x, y, z \in \{-1, 0, 1\}$
$S \leftarrow \infty$, $i \leftarrow \ell - 1$
while $i \geq 0$ do
    $D \leftarrow \{a, b, c\}, C \leftarrow 1$          /* D holds rows with reducible bits */
    foreach $k \in D$ do
        if $k_i = 0$ then remove $k$ from $D$
        else if $k_i + k_{i-2} = \pm 2$ then $C \leftarrow \textsc{max}(2, C)$
        else if $k_i + k_{i-3} = \pm 2$ then $C \leftarrow \textsc{max}(3, C)$
        else $D \leftarrow \emptyset$, $C \leftarrow 1$
    end
    for $j \leftarrow 1$ to $C - 1$ do
        if $a_{i-j} = b_{i-j} = c_{i-j} = 0$ then $D \leftarrow \emptyset$, $C \leftarrow 1$
    end
    foreach $k \in D$ do
        if $k_{i-2} \neq 0$ then $k_{i-1} \leftarrow k_i, k_{i-2} \leftarrow -k_i, k_i \leftarrow 0$
        else $k_{i-2} \leftarrow -k_i, k_{i-3} \leftarrow -k_i, k_i \leftarrow 0$
    end
    while $C > 0$ do
        $S \leftarrow \tau S$
        $S \leftarrow S + (a_i P + b_i Q + c_i R)$
        $i \leftarrow i - 1$ , $C \leftarrow C - 1$
    end
end
return $S$
```

curve estimates are for any generalized JSF.

Curve	Method	Precomp	A	D	M	Gain
P-192	NAF (separate)	0	194	576	6309	
P-192	JSF (simul)	10	123	192	2719	**56.9 %**
B-163	NAF (separate)	0	165	489	3276	
B-163	JSF (simul)	10	106	163	1500	**54.2 %**
K-163	NAF (separate)	0	165	-	1320	
K-163	JSF (simul)	10	106	-	848	**35.75 %**

Table 4.2: Elliptic curve operations needed for common curves.

The number of field multiplications is an estimate based on using mixed coordinates. For binary curves, the costs are given in Section 3.2 where $\mathsf{S} = 0$. For prime curves using affine and Jacobian coordinates, $\mathsf{A} = 8\mathsf{M} + 3\mathsf{S}$ and $\mathsf{D} = 4\mathsf{M} + 4\mathsf{S}$, where $\mathsf{S} = 0.85\mathsf{M}$. These are generally considered the most efficient methods [HMV04].

These figures show that when using simultaneous scalar multiplication the number of point additions and doublings (and hence the number of field multiplications) is significantly reduced due to the lower joint weight. While the efficiency gains are less when using Koblitz curves, the total amount of field multiplications are still less and therefore the most efficient.

4.4 Improving the Security of SCID Signatures

In Equation 4.4, one of the major disadvantages is that TTP escrows the private key to Alice. This not only requires a secure channel, but also means that both TTP and Alice have knowledge of Alice's private key s_A, so TTP can freely impersonate Alice. Avoiding this key escrow is desirable [PH97].

A solution was suggested in [AdM04] to avoid this disadvantage. It was written for multiplicative groups, where a proof of knowledge is needed to prevent users from obtaining certificates for a different identity. The proof of knowledge has been omitted here, and the threat of impersonation attacks is discussed in detail in Section 4.4.1.

The following blind key issuing protocol for groups over elliptic curves avoids key escrow and reaches Trust Level 3. It works in a way such that both TTP and Alice contribute to the randomness, yet only Alice has access to the final private key s_A.

Keygen. To generate a private key on user Alice's identity ID_A, Alice calculates

$$k_A G \text{ , where } k_A \in_R \mathbb{Z}_r^*$$

and sends $k_A G$ to TTP[2]. TTP calculates

$$\begin{aligned} (\bar{r}_A, b_A) &= \text{COMPRESS}(k_A G + k_T G) \text{ , where } k_T \in_R \mathbb{Z}_r^* \\ r_A &= \bar{r}_A + \text{H}(ID_A) \\ \bar{s}_A &= k_T - r_A s_D \pmod{r} \end{aligned} \tag{4.7}$$

[2]This point can be compressed if needed.

and sends $(r_A, b_A), \bar{s}_A$ to Alice. Alice calculates

$$s_A = k_A + \bar{s}_A \pmod{r} \tag{4.8}$$

Alice's public key is $W_A = s_A G$.

Correctness. The extracted public key is correct ($W_A = s_A G$):

$$\begin{aligned} W_A &= \text{DECOMPRESS}(r_A - \text{H}(ID_A), b_A) - r_A W_D \\ &= \text{DECOMPRESS}(\bar{r}_A + \text{H}(ID_A) - \text{H}(ID_A), b_A) - r_A W_D \\ &= k_A G + k_T G - r_A s_D G = (k_A + k_T - r_A s_D)G \\ &= (k_A + \bar{s}_A)G = s_A G \end{aligned}$$

4.4.1 The Threat of Impersonation Attacks

Using Equation 4.7, a user obtains a signature on it's identity from the trusted third party using random contributions from both the user and the trusted third party. While Equation 4.7 assumes that users are indeed generating random values, a user could possibly be attempting to obtain a key pair for another user's identity. Using multiplicative groups, Equation 4.7 would use the following operations.

$$\begin{aligned} \text{Alice: } g^{k_A} \text{ , where } k_A \in_R \mathbb{Z}_r &\Longrightarrow \text{ TTP} \\ \text{TTP: } g^{k_T} g^{k_A} \text{H}(ID_A) \text{ , where } k_T \in_R \mathbb{Z}_r &\Longrightarrow \text{ Alice} \end{aligned}$$

Now consider a user Malice who wants to obtain a valid signature from the trusted third party for Alice's identity as opposed to her own (*impersonation*). Malice chooses

$$\frac{g^{k_M}\text{H}(ID_A)}{\text{H}(ID_M)} \text{ , where } k_M \in_R \mathbb{Z}_r \Longrightarrow \text{TTP.}$$

The trusted third party then calculates

$$\begin{aligned} &\frac{g^{k_T}\text{H}(ID_M) g^{k_M}\text{H}(ID_A)}{\text{H}(ID_M)} \text{ , where } k_T \in_R \mathbb{Z}_r \\ &= g^{k_T} g^{k_M}\text{H}(ID_A) \Longrightarrow \text{Malice.} \end{aligned}$$

Thus, Malice has obtained a signature on Alice's identity from the trusted third party and can freely impersonate Alice.

While this is a serious security flaw for such cryptosystems that use multiplicative groups, impersonation turns out to be a much more difficult task when using groups over elliptic curves. Consider the above example where Malice wishes to obtain a signature on Alice's identity from the trusted third party. Using the protocol in Equation 4.7 over elliptic curves, to be successful in the same type of impersonation attack Malice needs to choose k_M and $k_M + d$ such that

$$\mathrm{H}(ID_M) + [(k_M + d)G_D + k_T G_D]_x = \mathrm{H}(ID_A) + [(k_M G_D + k_T G_D]_x \ .$$

Letting k denote $k_M + k_T$ (since k_T is random, so is k), this becomes

$$[(k + d)G_D]_x - [kG_D]_x = \mathrm{H}(ID_A) - \mathrm{H}(ID_M) \ . \tag{4.9}$$

One way to approach this problem is to estimate the probability that Equation 4.9 holds for a chosen d value, or formally

$$Prob_p\{[dG_D + P]_x - [P]_x = \mathrm{H}(ID_A) - \mathrm{H}(ID_M)\} \ .$$

This probability is upper bounded by its maximum value taken over all values of $\Delta x \in \mathbb{F}_q$ and non-zero differences $\Delta k \in \{1, 2, \ldots, \#E - 2\}$:

$$Prob_p\{[(\Delta k)G_D + P]_x - [P]_x = \Delta x\}$$

Example. Consider the elliptic curve

$$E : y^2 = x^3 + 26x + 3 \ , \ \mathbb{Z}_{31} \ , \ \#E = 33 \ , \ G_D = (2, 1) \ .$$

The 33 points on this curve are listed in Table 4.3. The x-coordinates of these points are symmetric around $k = \#E/2 = 16.5$.

Working modulo 31, the differences are obtained in the x-coordinate when varying k by $1, 2, \ldots, \#E - 2 = 31$. So for $k = 1, 2$, the following differences are computed.

$$k = 1\{2 - 16 = 17, 16 - 22 = 25, 22 - 7 = 15, \ldots, 16 - 2 = 14\} \pmod{31}$$
$$k = 2\{2 - 22 = 11, 16 - 7 = 9, 22 - 11 = 11, \ldots, 2 - 2 = 0\} \pmod{31}$$

Continuing through $k = \#E - 2 = 31$ (varying k by $\#E - 1$ yields ∞). The frequencies of the differences for each value of k are then totaled, to produce the frequencies listed in Table 4.4. Since the x-coordinates on E are symmetric (for prime fields), the difference frequencies are also symmetric around the same value.

k		k		k	
1	(2,1)	12	(21,13)	23	(19,3)
2	(16,12)	13	(9,6)	24	(29,25)
3	(22,1)	14	(30,10)	25	(5,17)
4	(7,30)	15	(8,17)	26	(18,17)
5	(11,15)	16	(4,4)	27	(12,20)
6	(12,11)	17	(4,27)	28	(11,16)
7	(18,14)	18	(8,14)	29	(7,1)
8	(5,14)	19	(30,21)	30	(22,30)
9	(29,6)	20	(9,25)	31	(16,19)
10	(19,28)	21	(21,18)	32	(2,30)
11	(24,6)	22	(24,25)	33	∞

Table 4.3: A small elliptic curve.

The larger the values of the frequency counts in comparison to the other frequency counts, the more chance of success there is for an impersonation attack. The maximum number of times a difference occurred was 4, meaning that Malice has about a 12% chance of success at such an impersonation attack for this (extremely) small curve (5 bits). The same experiment was also run on the 12-bit curve

$$E = y^2 = x^3 + 1347 \ , \ \mathbb{Z}_{3271} \ , \ \#E = 3373 \ , \ G_D = (544, 430) \ .$$

The maximum value in the resulting 3373 x 3271 table was also 4, meaning that Malice has about a 0.12% chance of success at such an impersonation attack for this (also extremely) small curve (12 bits). To put this figure in perspective, Malice has about a 0.03% chance of guessing a private key on this curve given only one attempt. These experimental results suggest that as the size of the curve increases, the probability of success of such an impersonation attack shrinks to an insignificant amount.

Δx	$\Delta k \in \{1, 2, \ldots, 31\}$
0	1111111111111111111111111111111
1	1110200121200111111002121002011
2	0101100220110122221011022001101
3	1000104113024210012420311401000
4	2102100222020000000020222001201
5	1341101002011100001110200101143
6	2021111200102111111201002111120
7	1201010000240212212042000010102
8	0010100012212022220212210001010
9	1202141001012010010210100141202
10	2020122121100101101001121221020
11	0313220211200001100002112022313
12	1010111110111411114111011111010
13	1011021101102212212201101120110
14	1121201121220042240022121102121
15	1102023110001000000100011320201
16	1102023110001000000100011320201
17	1121201121220042240022121102121
18	1011021101102212212201101120110
19	1010111110111411114111011111010
20	0313220211200001100002112022313
21	2020122121100101101001121221020
22	1202141001012010010210100141202
23	0010100012212022220212210001010
24	1201010000240212212042000010102
25	2021111200102111111201002111120
26	1341101002011100001110200101143
27	2102100222020000000020222001201
28	1000104113024210012420311401000
29	0101100220110122221011022001101
30	1110200121200111111002121002011

Table 4.4: Distribution of difference frequencies.

Chapter 5

Conclusions

This thesis has provided many efficient algorithms for elliptic curve cryptography. These results and contributions are widely applicable, from low-weight τ-adic joint representations to efficient signature verifications and secure key issuing protocols. The research has resulted in a few peer-reviewed publications.

1. A paper presenting a method for efficient self-certified identity-based signature verifications (Equation 4.6) was accepted for publication [Bru06a].

2. A paper presenting the generalized algorithm for generating a low-weight signed-bit τ-adic joint representation of an arbitrary number of integers (Algorithm 12) was accepted for publication [Bru06b].

3. A paper presenting a secure key issuing protocol is forthcoming (Sect. 4.4).

Future Work

The question of optimality and Algorithm 12 has not been explored. It is possible that there are methods that will lead to lower joint weights. More research is planned.

When it comes to compact digital signatures, probably the most active area of research is *paring-based cryptography* [BLS01]. Unfortunately, pairings are generally considered much more computationally difficult to compute then elliptic scalar multiplication. Efficient methods and settings for computing pairings could be future work in this area.

Bibliography

[AdM04] Giuseppe Ateniese and Breno de Medeiros. A provably secure Nyberg-Rueppel signature variant with applications. Cryptology ePrint Archive, Report 2004/093, 2004. http://eprint.iacr.org/.

[ADMRK02] Essame Al-Daoud, Ramlan Mahmod, Mohammad Rushdan, and Adem Kilicman. A new addition formula for elliptic curves over GF(2^n). *IEEE Trans. Comput.*, 51(8):972–975, 2002.

[ANS98] ANSI. The elliptic curve digital signature algorithm. Technical Report ANSI X9.62-1998, American National Standards Institute, September 1998.

[Ari43] Aristotle. *Aristotle's Politics.* The Modern library, New York, NY, 1943. Translated by Benjamin Jowett.

[BBB$^+$05] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management - part 1: General. Technical Report NIST Special Publication 800-57, August 2005.

[BHLM01] Michael Brown, Darrel Hankerson, Julio López, and Alfred Menezes. Software implementation of the nist elliptic curves over prime fields. In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 250–265, London, UK, 2001. Springer-Verlag. extended version available as CORR 2000-56.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532, London, UK, 2001. Springer-Verlag.

[Bru06a] Billy Bob Brumley. Efficient three-term simultaneous elliptic scalar multiplication with applications. In Viiveke Fåk, editor, *Proceedings of the 11th Nordic Workshop on Secure IT Systems (NordSec 2006)*, pages 105–116, Linköping, Sweden, 2006.

[Bru06b] Billy Bob Brumley. Left-to-right signed-bit τ-adic representations of n integers (short paper). In *International Conference on Information and Communications Security – ICICS'06*, volume 4307 of *Lecture Notes in Computer Science*, pages 469–478, Raleigh, North Carolina, USA, December 2006. Springer-Verlag.

[BW05] S. Blake-Wilson. Ecc cipher suites for tls. Technical report, IETF, October 2005.

[Caj91] Florian A. Cajori. *A History of Mathematics*. Chelsea, New York, 5th edition, 1991.

[CC05] Hannu Kari Catharina Candolin, Janne Lundberg. Packet level authentication in military networks. In *Proceedings of the 6th Australian Information Warfare & IT Security Conference*, Geelong, Australia, November 2005.

[CLSQ03] Mathieu Ciet, Tanja Lange, Francesco Sica, and Jean-Jacques Quisquater. Improved algorithms for efficient arithmetic on elliptic curves using fast endomorphisms. *Advances in Cryptology-Eurocrypt 2003*, 2656:388–400, 2003.

[Dai06] Wei Dai. *Crypto++ Library*, 2006. Available at http://www.eskimo.com/~weidai/cryptlib.html.

[DHH$^+$04] Ricardo Dahab, Darrel Hankerson, Fei Hu, Men Long, Julio López, and Alfred Menezes. Software multiplication using normal bases. Technical Report CACR 2004-12, Centre for Applied Cryptographic Research, University of Waterloo, Canada, 2004.

[ElG85] Taher ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[FD04] Haining Fan and Yiqi Dai. Two software normal basis multiplication algorithms for gf(2n). Cryptology ePrint Archive, Report 2004/126, 2004. http://eprint.iacr.org/.

[GHP04] Peter J. Grabner, Clemens Heuberger, and Helmut Prodinger. Distribution results for low-weight binary representations for pairs of integers. *Theoretical Computer Science*, 319(1-3):307–331, June 2004.

[Gir91] Marc Girault. Self-certified public keys. In Donald W. Davies, editor, *Advances in Cryptology - EuroCrypt '91*, pages 490–497, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 547.

[Gor98] Daniel M. Gordon. A survey of fast exponentiation methods. *J. Algorithms*, 27(1):129–146, 1998.

[Har68] Garrett Hardin. The tragedy of the commons. *Science*, 162, 1968.

[HHM00] Darrel Hankerson, Julio López Hernandez, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. In *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–24, London, UK, 2000. Springer-Verlag.

[HMV04] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer, New York, 2004.

[IEE99] IEEE. Standard specifications for public-key cryptography. Technical Report IEEE P1363 / D13, Institute of Electrical and Electronics Engineers, Inc., November 12 1999.

[IT88] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and Computing*, 78(3):171–177, September 1988.

[Knu98] Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, MA, 3rd edition, 1998.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Kob92] Neal Koblitz. Cm-curves with good cryptographic properties. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 279–287, London, UK, 1992. Springer-Verlag.

[Kob94] Neal Koblitz. *A Course in Number Theory and Cryptography.* Number 114 in Graduate Texts in Mathematics. Springer, 2nd edition, 1994.

[Kra93] David W. Kravitz. Us patent #5231668: Digital signature algorithm, 1993.

[Lan78] Serge Lang. *Elliptic Curves: Diophantine Analysis.* Springer, 1978.

[LD99] Julio López and Ricardo Dahab. Improved algorithms for elliptic curve arithmetic in GF(2^n). In *SAC '98: Proceedings of the Selected Areas in Cryptography*, pages 201–212, London, UK, 1999. Springer-Verlag.

[LG06] LiDIA-Group. *LiDIA - A C++ Library For Computational Number Theory.* The LiDIA Group, 2006. Available at http://www.informatik.tu-darmstadt.de/TI/LiDIA/.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In *CRYPTO '85: Advances in Cryptology*, pages 417–426, London, UK, 1986. Springer-Verlag.

[MO90] Morain and Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *RAIRO: R. A. I. R. O. Informatique Theorique et Applications/Theoretical Informatics and Applications*, 24, 1990.

[MS92] W. Meier and O. Staffelbach. Efficient multiplication on certain nonsupersingular elliptic curves. In Ernest F. Brickell, editor, *Advances in Cryptology - Crypto '92*, pages 333–344, Berlin, 1992. Springer-Verlag. Lecture Notes in Computer Science Volume 740.

[New56] James R. Newman, editor. *The World of Mathematics. Four volumes.* Simon and Schuster, New York, 1956.

[NIS99] NIST. Recommended elliptic curves for federal government use. National Institute of Standards and Technology, May 1999.

[NR93] Kaisa Nyberg and Rainer A. Rueppel. A new signature scheme based on the dsa giving message recovery. In *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, pages 58–61, New York, NY, USA, 1993. ACM Press.

[NY01] Peng Ning and Yiqun Lisa Yin. Efficient software implementation for finite field multiplication in normal basis. In *ICICS '01: Proceedings of the Third International Conference on Information and Communications Security*, pages 177–188, London, UK, 2001. Springer-Verlag.

[PH97] H. Petersen and P. Horster. Self-Certified Keys: Concepts and Applications. In *Proceedings of the Third International Conference on Communications and Multimedia Security*, pages 102–116, 1997.

[Pro03] John Proos. Joint sparse forms and generating zero columns when combing. Technical Report CORR 2003-23, Centre for Applied Cryptographic Research, University of Waterloo, Canada, 2003.

[RK04] Xiaoyu Ruan and Rajendra S. Katti. Low-weight left-to-right binary signed-digit representation of n integers. In *2004 IEEE International Symposium on Information Theory*, June 2004.

[RM06] Arash Reyhani-Masoleh. Efficient algorithms and architectures for field multiplication using gaussian normal bases. *IEEE Transactions on Computers*, 55(1):34–47, January 2006.

[RMH01] Arash Reyhani-Masoleh and M. Anwar Hasan. Fast normal basis multiplication using general purpose processors. Technical Report CORR 2001-25, Centre for Applied Cryptographic Research, University of Waterloo, Canada, 2001.

[RMH03] Arash Reyhani-Masoleh and M. Anwar Hasan. Fast normal basis multiplication using general purpose processors. *IEEE Transactions on Computers*, 52(11):1379–1390, November 2003.

[Ros99] Michael Rosing. *Implementing elliptic curve cryptography.* Manning Publications Co., Greenwich, CT, 1999.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sch91] C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sco06] Michael Scott. *MIRACL—A Multiprecision Integer and Rational Arithmetic C/C++ Library*. Shamus Software Ltd, Dublin, Ireland, 2006. Available at `http://indigo.ie/~mscott`.

[SEC00] SECG. Standards for efficient cryptography. Technical Report Version 1.0, Standards for Efficient Cryptography Group, September 20 2000.

[Sol00] Jerome A. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes, and Cryptography*, 19(2–3):195–249, March 2000.

[Sol01] Jerome A. Solinas. Low-weight binary representations for pairs of integers. Technical Report CORR 2001-41, Centre for Applied Cryptographic Research, University of Waterloo, Canada, 2001.

[Str64] Ernst G. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 71:806–808, 1964.