Des. Codes Cryptogr. (2017) 83:565–588
DOI 10.1007/s10623-016-0251-2

Multiple point compression on elliptic curves

Xinxin Fan[1] · **Adilet Otemissov**[2] · **Francesco Sica**[3] · **Andrey Sidorenko**[4]

Received: 27 August 2015 / Revised: 24 June 2016 / Accepted: 7 July 2016 /
Published online: 25 July 2016

Abstract Point compression is an essential technique to save bandwidth and memory when deploying elliptic curve based security solutions in wireless communication systems. In this contribution, we provide new linear algebra (LA) based compression algorithms for multiple points on elliptic curves, that are compression algorithms which only make use of LA (with a constant number of field multiplications and at most one inversion, with no quadratic or higher degree polynomial root finding). In particular, we extend the results of Khabbazian et al. (IEEE Trans Comput 56(3):305–313, 2007) to four (resp. five) points on elliptic curves by generically storing five (resp. six) field elements and provide an asymptotic generalization to any number n of points on a curve $y^2 = f(x)$ by generically storing $n + 1$ values.

Keywords Cryptography · Elliptic curves · Point compression

Mathematics Subject Classification 11T71 · 94A60

Communicated by C. Mitchell.

This work was done when Xinxin Fan was a research associate at the University of Waterloo. Adilet Otemissov worked on part of this project as his capstone thesis at Nazarbayev University.

✉ Francesco Sica
francesco.sica@nu.edu.kz

Xinxin Fan
Xinxin.Fan@us.bosch.com

Adilet Otemissov
adilet.otemissov@postgrad.manchester.ac.uk

Andrey Sidorenko
sidorenko@brightsight.com

1 Robert Bosch LLC, Bosch Research and Technology Center, Pittsburgh, PA 15222, USA

2 University of Manchester, Manchester, UK

3 School of Science and Technology, Nazarbayev University, 53 Kabanbay Batyr Avenue, 010000 Astana, Kazakhstan

4 Brightsight, Delftechpark 1, 2628XJ Delft, The Netherlands

1 Introduction

The rapid progress in wireless communication systems, personal communication systems, and smartcard technologies is impacting people's life at a staggering rate and in significant ways. New security challenges have to be solved in order to protect those communication systems from malicious attacks. Elliptic curve cryptosystems (ECC) provide efficient and robust solutions to many of the existing security issues in wireless communication systems [9,10,32]. When implementing ECC based cryptographic protocols on smart devices such as RFID tags, smart cards, and wireless sensor nodes, it is often necessary to transmit or store points on elliptic curves. Since those smart devices usually have constrained resources in terms of computational capabilities, memory, and bandwidth, it is desirable to represent a point in a compact form.

Point compression is a technique that allows points on an elliptic curve to be represented with fewer bits of data, thereby reducing both the storage space on smart devices as well as the amount of data that needs to be transmitted to and from those devices [5]. A single point compression usually involves solving a quadratic equation over a finite field, which might be computationally expensive for low-cost smart devices. To overcome the aforementioned issue, Khabbazian et al. [20] proposed a double and a triple point compression schemes, respectively, which allow a compact representation of two or three points on an elliptic curve without the computational cost associated with ordinary single point compression. Moreover, the authors pointed out that the point compression can be used to reduce the required bandwidth in the case of the random point multiplication. We notice that double point compression technique can also be directly applied to some pairing-free identity-based signature schemes (see [8,17] for example) to reduce the required bandwidth during wireless transmission. Additional applications of various point compression techniques include the following.

Message authentication in vehicular ad hoc networks (VANETs)

In VANETs, vehicles are equipped with Dedicated Short Range Communication (DSRC) capabilities to enable vehicles' On-Board Units (OBUs) communicate wirelessly with other vehicles' OBU or Road Side Units (RSUs) [26]. Safety provision is one of the core goals of VANET deployment. To this end, an OBU periodically broadcasts traffic-related messages to surrounding vehicles in an authenticated manner. Elliptic curve cryptography and digital certificates have been employed to ensure the authenticity of broadcast safety messages [19]. To save the communication bandwidth in VANETs, public keys in digital certificates are stored and transmitted in compressed form. For verifying digital signatures, an OBU needs first to decompress the received public keys, followed by the verification of multiple digital signatures. Multiple point compression can be directly applied in this case to accelerate the point decompression for multiple public keys, which provides an efficient method to achieving a good trade-off between decompression performance and storage requirement.

Multi-user broadcast authentication in wireless sensor networks (WSNs)

Multi-user broadcast is an efficient and common communication paradigm in WSNs, where multiple users disseminate messages (i.e., queries or commands) into networks for retrieving the information of their interest [25,34]. Due to the constrained resources of wireless sensor nodes, elliptic curve cryptography and digital certificates are usually used to ensure the authenticity of the broadcast messages in WSNs [15]. Again, public keys of network users are stored in digital certificates in compressed form to save the energy consumption of sensor nodes. When receiving broadcast messages from multiple

users, sensor nodes will verify both digital certificates and signatures. Application of multiple point compression technique enables sensor nodes to balance the performance for signature verifications and the memory footprint on wireless sensor nodes.

Countermeasures against attacks

Nowadays ECDSA [3], ECDH [4] and other elliptic-curve based cryptographic algorithms are often implemented on secure devices such as smartcards. Such devices must be protected against side channel attacks and perturbation attacks.
A well-known countermeasure against many side channel attacks is to use randomized projective coordinates. If a smartcard stores only x-coordinates of several points and the sum of their y-coordinates it is possible to randomize or to re-randomize the coordinates in an efficient manner with less field multiplications. The whole set of points will get a common randomly chosen z-coordinate.
A classical countermeasure against perturbation attacks is the integrity check, such as a parity check or a CRC check. In the case when elliptic-curve points are compressed they can be stored in a smaller buffer and thus the checksum or the CRC value can be computed more efficiently [29].

Other potential applications

The ciphertext of the ElGamal Elliptic Curve Cryptosystem [21] is a pair of points of an elliptic curve. Point compression may result in shorter ciphertexts, which is crucial for many applications.
The internal state of some ECC-based pseudorandom generators includes two points of an elliptic curve. If the pseudorandom generator is rarely invoked it makes sense to store its state in a compressed form and decompress it when needed. In particular, the two-point compression method can be effective. An example of such a pseudorandom generator is `Dual_EC_DRBG` [27]. This pseudorandom generator has recently been deprecated by NIST due to certain security flaws [28]. The point compression techniques are likely to be applicable to the successors of `Dual_EC_DRBG` as well.

Considering potential applications of multiple point compression technique, we present new linear algebra (LA) based compression algorithms for multiple points on elliptic curves in this paper. Let us emphasize from the start that the goal of these new compression algorithms is not to reduce storage requirements by achieving a higher compression rate, but rather to trade current compression techniques with one which is asymptotically (in storing many points) almost as good and additionally achieves a superior decompression speed.

To this end, a four- and five-point compression scheme is proposed which further generalizes the Khabbazian et al. [20]. Our new compression algorithms only make use of basic linear algebra (multiplications, addition, subtractions and at most one division) and do not need to solve any quadratic or higher degree equations over finite fields. In particular, the number of equivalent basic field multiplications (without considering the division/inversion) is constant and does not increase with the field size. Let us note that the running time for one division is equivalent to finding one square root. However, the strength of our method is that k inversions can be combined into a single inversion and $3k - 3$ multiplications [11, Algorithm 10.3.4 p. 489 a.k.a. Montgomery's trick], whereas several square root extractions cannot be combined into one modulo a constant number of multiplications.

The remainder of this paper is organized as follows. Section 2 briefly reviews the previous work for double and triple point compression schemes in [20], followed by the new four and five point LA based compression algorithm in Sect. 3. In Sect. 4, we generalize this method to an arbitrary number of points. Finally, Sect. 5 concludes this work.

2 Compression algorithms and previous work

For simplicity in this work we will restrict ourselves to curves over odd characteristic fields $\mathbb{F}$. Given a plane equation $y^2 = f(x)$, where f is a polynomial in $\mathbb{F}[x]$ without multiple roots, we usually store a point $P_1 = (x_1, y_1)$ by its coordinates (so called trivial, or no compression), otherwise by x_1 plus one extra bit and recovering y_1 by solving a quadratic equation in $\mathbb{F}$, using the bit to discriminate between the two solutions. The latter method requires less memory, but more resources to decompress the point in order to find all its coordinates. In particular solving a quadratic equation in $\mathbb{F}$ requires the equivalent of a field inversion, costing a number of field multiplications which increases linearly in the field bit size. For instance, solving $y^2 \equiv a \pmod{p}$ for $p \equiv 3 \pmod{4}$ will entail computing $a^{(p+1)/4} \mod p$.

We are interested in this work to study compression-decompression algorithms which make use of a *constant* number of field operations (multiplications, additions and inversions, with inversions grouped into a single one), independently of the field size. These methods will typically make use of linear algebra only, therefore we are going to call them LA methods. We can distinguish between the following two flavours of compression-decompression algorithms:

(1) LA compression, non-LA decompression (e.g. the classical method of storing x_1 and 1 bit, recover y_1 by solving a quadratic equation),
(2) LA compression, LA decompression (fully LA).

Additionally, one could conceive of a non-LA compression, LA decompression algorithm, although we could not find any in the literature. Nevertheless, our new compression method below (see Sect. 2.2) for two points is a first step towards this paradigm of a slow compression and very fast decompression method.

In [20] a fully LA method is presented to compress $P_1, \ldots, P_n$ points when $n = 2, 3$. The object of the present article is to extend their work to all values of n less than the characteristic of the field and to present an efficient implementation of this when $n = 4, 5$.

We review the results of [20] here, in the case of fields of characteristic at least 5. Minor modifications can adapt this method to characteristic 2 and 3. Note that the exact performance improvement of multiple point compression technique over the traditional approach highly depends on the efficiency of solving a quadratic equation over finite fields (see Sect. 3.4 for more details).

2.1 Two-point compression

Given an elliptic curve $y^2 = x^3 + ax + b = f(x)$ and points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, we can represent them in general by $(x_1, x_2, \alpha = y_1 + y_2)$. Then

$$y_1 = \frac{\alpha}{2} + \frac{x_1 - x_2}{8\alpha}\left(3(x_1 + x_2)^2 + (x_1 - x_2)^2 + 4a\right)$$

and

$$y_2 = \alpha - y_1 \ .$$

This is easily seen by remarking that

$$y_1 - y_2 = \frac{y_1^2 - y_2^2}{y_1 + y_2} = \frac{f(x_1) - f(x_2)}{\alpha}$$

and knowledge of $\alpha = y_1 + y_2$ will readily yield y_1, y_2 by linear algebra. This algorithm works, as long as $\alpha \neq 0$. This condition can be checked a priori before decompression by computing $y_i^2 = f(x_i)$ and checking that $y_1^2 \neq y_2^2$ (a stronger condition). Otherwise, if $y_1^2 = y_2^2$, the compression algorithm stores (x_1, x_2, y_1, b) where b is an extra bit to distinguish between the generic case and the exceptional case when $y_2 = -y_1$. Two-point compression can thus be achieved by storing 3 field elements and up to one extra bit.[1]

When $\alpha \neq 0$, the computational complexity of the decompression is $2M + 2S + I$.

2.2 Alternative two-point compression

We introduce new two-point compression algorithm which requires an inverse operation in compression part while decompression phase needs only multiplication and square operations.

In this case, points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are represented by $\left(A = \frac{x_1 - x_2}{y_1 - y_2}, B = y_1 - y_2, x_2\right)$. Decompression algorithm works as follows:

Since $x_1 - x_2 = AB$ we obtain $x_1 = AB + x_2$.

Then, having two elliptic curve equations

$$y_1^2 = x_1^3 + ax_1 + b$$
$$y_2^2 = x_2^3 + ax_2 + b$$

the following two expressions are derived:

$$y_1^2 - y_2^2 = x_1^3 - x_2^3 + a(x_1 - x_2)$$
$$y_1 + y_2 = \frac{x_1 - x_2}{y_1 - y_2}\left((x_1 - x_2)^2 + 3x_1x_2 + a\right) = A\left((x_1 - x_2)^2 + 3x_1x_2 + a\right)$$

Since $y_1 - y_2 = B$, y_1 and y_2 are easily obtained by Linear Algebra:

$$y_1 = \frac{A\left((x_1 - x_2)^2 + 3x_1x_2 + a\right) + B}{2}$$
$$y_2 = y_1 - B$$

The overall computational cost of the algorithm is $3M+1S$ if multiplication by constant elements is disregarded. This algorithm does require an inverse operation in compression phase. However, decompression part needs only multiplication and square operations, which considerably reduces its computational complexity compared to previously described two-point decompression method.

The algorithm works when $y_1^2 \neq y_2^2$. In case $y_1^2 = y_2^2$, follow the compression algorithm described in the Sect. 2.1.

2.3 Three-point compression

In this case, the generic compression algorithm represents $P_i = (x_i, y_i)$ by $(x_1, x_2, x_3, \alpha = y_1 + y_2 + y_3)$. Decompression works as follows. Let

$$\beta = \alpha^2 + y_3^2 - y_1^2 - y_2^2 = 2(y_3 + y_1)(y_3 + y_2).$$

[1] In fact, as noticed by a reviewer of a previous version of this work, since one inversion is roughly equivalent to one square root extraction, there is the easier compression method of always storing (x_1, x_2, y_1, b), where b is a bit allowing to compute y_2 from $f(x_2) = y_2^2$. The decompression then simply costs 1 inversion.

Then

$$y_3 = \frac{\beta^2 + 4\alpha^2 y_3^2 - 4y_1^2 y_2^2}{4\alpha\beta}$$

and since $y_1 + y_2 = \alpha - y_3$ we recover y_1, y_2 by the previous two-point compression algorithm. We are in the generic case provided $\alpha \neq 0$ and $y_i^2 \neq y_j^2$ for $1 \leq i < j \leq 3$, in which case by the above $\beta \neq 0$. We look at the exceptional cases:

(1) $y_i^2 \neq y_j^2$ for $1 \leq i < j \leq 3$ and $\alpha = 0$. In this case, the compression part will store $(x_1, x_2, x_3, y_1 + y_2)$. We can recover y_1, y_2 by the generic case of two-point compression and then $y_3 = -y_1 - y_2$.
(2) Two squares are equal but not three. Without loss of generality, $y_1^2 = y_2^2 \neq y_3^2$. In this case, represent the three points as $(x_1, x_2, x_3, y_2 + y_3, q_1)$, where $q_1 \in \{0, 1\}$. Decompression works out y_2, y_3 using generic two-point compression, and bit q_1 is used to tell if $y_1 = y_2$ or $-y_2$.
(3) $y_1^2 = y_2^2 = y_3^2$. In this case, store $(x_1, x_2, x_3, y_1, q_1, q_2)$, with $b_i \in \{0, 1\}$. The bits are used to pinpoint the sign of y_2 and y_3 compared to y_1.

Three-point compression can thus be achieved by storing 4 field elements and up to 2 extra bits. As shown in [20], in the computational complexity in the generic case is $12M + 6S + I$.

We next present our generalization of this method to four points.

3 Four- and five-point compression

3.1 General idea

Let n be the number of points to be compressed and let $y_1, y_2, \ldots, y_n$ be the y-coordinates of these points. Denote

$$\begin{aligned} a_1 &= y_1 + y_2 + \cdots + y_n, \\ a_2 &= \sum_{1 \leq i < j \leq n} y_i y_j, \\ &\cdots \\ a_n &= y_1 y_2 \ldots y_{n-1} y_n \end{aligned}$$

the elementary symmetric polynomials in the y_i's. We want to express y_i in terms of the a_j's and the even powers of y_i, since we know the value $y_i^2 = f(x_i)$.

Theorem 1 *For even values of n the y-coordinates $y_1, \ldots, y_n$ can be generically expressed as follows:*

$$y_i = \frac{a_n + y_i^2 a_{n-2} + \cdots + y_i^{n-2} a_2 + y_i^n}{a_{n-1} + y_i^2 a_{n-3} + \cdots + y_i^{n-2} a_1}, \quad i = 1, \ldots, n. \tag{1}$$

For odd values of n,

$$y_i = \frac{a_n + y_i^2 a_{n-2} + \cdots + y_i^{n-1} a_1}{a_{n-1} + y_i^2 a_{n-3} + \cdots + y_i^{n-3} a_2 + y_i^{n-1}}, \; i = 1, \ldots, n. \tag{2}$$

Proof We have

$$(x + y_1) \cdots (x + y_n) = x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n \; .$$

Letting $x = -y_i$, we get

$$\begin{aligned} 0 &= a_n - y_i a_{n-1} + y_i^2 a_{n-2} - y_i^3 a_{n-3} + \cdots \\ &= a_n + y_i^2 a_{n-2} + \cdots - y_i (a_{n-1} + y_i^2 a_{n-3} + \cdots) \end{aligned}$$

which is what we want. □

We will not consider the exceptional situations when the denominators are equal to 0 (see Sect. 4 for a discussion of this).

3.2 Four-point compression

As a next step, we introduce a four-point compression algorithm based on Theorem 1. The compression algorithm represents the four points $P_i, i = 1, 2, 3, 4$, by $(x_1, x_2, x_3, x_4, a_1 = y_1 + y_2 + y_3 + y_4)$. The decompression algorithm first retrieves a_2, a_3, a_4 and then uses the following equation to compute y_i's:

$$y_i = \frac{a_4 + y_i^2 a_2 + y_i^4}{a_3 + y_i^2 a_1}, \quad \text{for} \quad i = 1, 2, 3, 4. \tag{3}$$

Our initial goal is to express the elementary symmetric polynomials a_2, a_3 and a_4 in terms of a_1 and y_i^2's. Note that

$$a_2 = \frac{a_1^2 - b_1}{2}, \tag{4}$$

$$a_1 a_3 - a_4 = \frac{a_2^2 - b_2}{2}, \tag{5}$$

$$a_2 a_4 = \frac{a_3^2 - b_3}{2}, \tag{6}$$

where

$$\begin{aligned} b_1 &= y_1^2 + y_2^2 + y_3^2 + y_4^2, \\ b_2 &= y_1^2 y_2^2 + y_1^2 y_3^2 + y_1^2 y_4^2 + y_2^2 y_3^2 + y_2^2 y_4^2 + y_3^2 y_4^2, \\ b_3 &= y_1^2 y_2^2 y_3^2 + y_1^2 y_2^2 y_4^2 + y_1^2 y_3^2 y_4^2 + y_2^2 y_3^2 y_4^2. \end{aligned}$$

From (4) a_2 is easily computable. Denote the right-hand side of Eq. (5) as A. A is computable since a_2 is known. Hence, from (5) we get the following linear relationship between a_3 and a_4:

$$a_3 = \frac{A + a_4}{a_1}. \tag{7}$$

Substituting a_3 in Eq. (6) by the above expression gives the following quadratic equation in a_4:

$$2a_1^2 a_2 a_4 = A^2 + 2Aa_4 + a_4^2 - a_1^2 b_3.$$

a_4^2 is efficiently computable since it is equal to $y_1^2 y_2^2 y_3^2 y_4^2$. Therefore

$$a_4 = \frac{A^2 + a_4^2 - a_1^2 b_3}{2(a_1^2 a_2 - A)}. \tag{8}$$

Formula (7) implies that

$$a_3 = \frac{2a_1^2a_2A - A^2 + a_4^2 - a_1^2b_3}{2a_1\left(a_1^2a_2 - A\right)}. \tag{9}$$

Substituting a_3 and a_4 from (8) and (9) into (3) gives

$$y_i = \frac{a_1\left(A^2 + a_4^2 - a_1^2b_3 + 2\left(a_1^2a_2 - A\right)\left(y_i^2a_2 + y_i^4\right)\right)}{2a_1^2a_2A - A^2 + a_4^2 - a_1^2b_3 + 2a_1^2y_i^2\left(a_1^2a_2 - A\right)}. \tag{10}$$

The computational complexity of the decompression algorithm described above is $55M + 11S + 1I$, where M, S, and I denote the complexity of the final field multiplication, squaring and inversion, respectively. The complexity can be reduced significantly when only one y-coordinate has to be retrieved. For more details, the reader is referred to Appendix 2. The calculation of the computational complexity is based on the principles described in [23].

3.3 Five-point compression

Theorem 1 can also be used as a basis for a five-point compression algorithm. The compression algorithm represents the five points $P_i, i = 1, 2, 3, 4, 5$, by $(x_1, x_2, x_3, x_4, x_5, a_1 = y_1 + y_2 + y_3 + y_4 + y_5)$. The decompression algorithm exploits Eq. (2) for $n = 5$:

$$a_2 = (a_1^2 - b_1)/2, \tag{11}$$

$$a_3a_1 - a_4 = (a_2^2 - b_2)/2, \tag{12}$$

$$a_4a_2 - a_5a_1 = (a_3^2 - b_3)/2, \tag{13}$$

$$a_5a_3 = (a_4^2 - b_4)/2. \tag{14}$$

Equation (11) immediately yields a_2 since a_1 and b_i, $i = 1, 2, 3, 4$, are known. It follows from (12) that a_3 can be expressed as a linear function of a_4:

$$a_3 = (A + a_4)/a_1. \tag{15}$$

Here Formula (15) substituted into (13) gives

$$a_4a_2 - a_5a_1 = \left(\frac{A^2 + 2Aa_4 + a_4^2}{a_1^2} - b_3\right)\Big/2,$$

which can be used to express a_4^2 as follows:

$$a_4^2 = a_1^2(2a_4a_2 - 2a_5a_1 + b_3) - A^2 - 2Aa_4. \tag{16}$$

As a next step, we use this expression as well as (15) to rewrite (14) as follows:

$$a_5 \cdot \left(\frac{A + a_4}{a_1}\right) = \left[a_1^2\left(2a_4a_2 - 2a_5a_1 + b_3\right) - A^2 - 2Aa_4 - b_4\right]/2,$$

which implies

$$a_4 = \frac{a_5\left(2a_1^4 + 2A\right) + a_1\left(A^2 + b_4 - b_3a_1^2\right)}{a_1\left(2a_1^2a_2 - 2A\right) - 2a_5}. \tag{17}$$

As before, let $C = a_1\left(2a_1^2a_2 - 2A\right)$. Let $F = 2a_1^4 + 2A$ and $G = a_1\left(A^2 + b_4 - b_3a_1^2\right)$. Then

$$a_4 = \frac{Fa_5 + G}{C - 2a_5}.$$

Now formula $a_4^2 = 2a_5a_3 + b_4$, which follows directly from (14), can be written as follows:

$$\left(\frac{Fa_5 + G}{C - 2a_5}\right)^2 = 2a_5 \cdot \left(\frac{A + a_4}{a_1}\right) + b_4$$

or

$$\left(\frac{Fa_5 + G}{C - 2a_5}\right)^2 = 2a_5 \cdot \left(\frac{A + (Fa_5 + G)/(C - 2a_5)}{a_1}\right) + b_4.$$

Note that a_5^2 can be efficiently computed, $a_5^2 = y_1^2y_2^2y_3^2y_4^2y_5^2$. Thus we get

$$a_5 = \frac{-F^2a_5^2a_1 - G^2a_1 - 8ACa_5^2 + 2FCa_5^2 - 4Ga_5^2 + C^2b_4a_1 + 4b_4a_1a_5^2}{2GFa_1 - 2AC^2 - 8Aa_5^2 - 2GC + 4Fa_5^2 + 4Cb_4a_1}.$$

Let J be the numerator and let H be the denominator of the formula so $a_5 = J/H$. Then

$$a_4 = \frac{FJ + GH}{CH - 2J}.$$

Denote $K = CH - 2J$, $L = FJ + GH$. Then

$$a_4 = L/K.$$

and

$$a_3 = \frac{AK + L}{a_1K}.$$

Since $n = 5$

$$y_i = \frac{a_5 + y_i^2a_3 + y_i^4a_1}{a_4 + y_i^2a_2 + y_i^4},$$

where $i = 1, 2, 3, 4, 5$. Using the expressions above for a_3, a_4, a_5 we get

$$y_i = \frac{JKa_1 + Hy_i^2\left(AK + L + Ka_1^2y_i^2\right)}{Ha_1\left(L + Ky_i^2(a_2 + y_i^2)\right)}. \tag{18}$$

Let M_i be the numerator of y_i: $M_i = JKa_1 + Hy_i^2(AK + L + Ka_1^2y_i^2)$. Then y_i can be converted to the common denominator

$$N = Ha_1\left(L + Ky_1^2\left(a_2 + y_1^2\right)\right) \cdot \left(L + Ky_2^2\left(a_2 + y_2^2\right)\right) \cdots \left(L + Ky_5^2\left(a_2 + y_5^2\right)\right)$$

as follows:

$$y_1 = \frac{M_1\left(L + Ky_2^2\left(a_2 + y_2^2\right)\right) \cdots \left(L + Ky_5^2\left(a_2 + y_5^2\right)\right)}{N},$$

$$\cdots$$

$$y_5 = \frac{M_5\left(L + Ky_1^2\left(a_2 + y_1^2\right)\right) \cdots \left(L + Ky_4^2\left(a_2 + y_4^2\right)\right)}{N}. \tag{19}$$

The latter formula can be used for the efficient computation of y_1, y_2, y_3, y_4, y_5.

The overall computational complexity of the decompression is $120M + 12S + I$ in the case when the goal is to retrieve the y-coordinates of all five points. As before, the complexity can be reduced if the goal of the algorithm is to retrieve the y-coordinate of only one point. Please refer to Appendix 3 for details.

3.4 Discussion on the occupied memory and computational complexity

The two-point and three-point compression algorithms proposed by [20] as well as the four-point and five-point compression algorithms presented in this paper reduce the computational cost of the decompression compared to the ordinary single point compression, in which only the x-coordinates are stored and the y-coordinates are extracted through square root operations. On the other hand, the multiple point compression algorithms require roughly $\frac{100}{n}\%$ more storage space than the ordinary point compression, where n denotes the number of points to compress.

3.4.1 Square-and-multiply exponentiation, $p \equiv 3 \pmod 4$

We will make a concrete comparison for the case of 256-bit curves. Observe that solving $x^2 \equiv a \pmod p$ for $p \equiv 3 \pmod 4$ translates into computing $a^{(p+1)/4} \mod p$. At the same time, $x^{-1} \pmod p$ can be computed as $x^{p-2} \pmod p$. Thus we may assume that the square root operation and the inversion operation require roughly the same number of operations on average, that is $R \approx I \approx 127M + 254S$. The complexity of the inversion operation can be reduced by applying the extended Euclidean algorithm instead of the exponentiation. For the sake of simplicity we do not consider this enhancement in the two tables below; in Sect. 3.4.5 alternative values of R and I are discussed.

The following table provides a comparison between the ordinary-point and multiple-point compression algorithms.

n	Occupied memory		Computational complexity	
	Ordinary	Multiple	Ordinary	Multiple
2	2	3	$2R = 254M + 508S$	$129M + 256S$
3	3	4	$3R = 381M + 762S$	$139M + 260S$
4	4	5	$4R = 508M + 1016S$	$182M + 265S$
5	5	6	$5R = 635M + 1270S$	$247M + 266S$

Here the unit of memory is one field element; columns "ordinary" and "multiple" correspond to the ordinary-point and multiple-point compression algorithms, respectively. In this table we neglect the extra storage bits needed in exceptional situations (the reader is referred to Sect. 4 for more details).

The table above can be used to illustrate the differences in the memory usage and in the computational complexity between the ordinary-point and multiple-point compression algorithms. We assume that $S = 0.8M$, which is one of the possibilities considered in the database [23]. Then the table can be rewritten as follows.

n	Occupied memory			Computational complexity		
	Ordinary	Multiple	Percentage	Ordinary	Multiple	Percentage
2	2	3	+50	$660M$	$334M$	−49
3	3	4	+33	$991M$	$347M$	−65
4	4	5	+25	$1321M$	$394M$	−70
5	5	6	+20	$1651M$	$460M$	−72

The new two-point compression algorithm described in Sect. 2.2 requires one inversion operation in its compression phase. However, the computational complexity of decompression part is $3M + 1S$ which requires 99.4 % less time than decompression phase of ordinary point compression method.

Please note that the multi-exponentiation technique [33] cannot be used to speed up the ordinary point compression algorithm. The end result of the multi-exponentiation is a product of several exponentiations while the ordinary point compression needs the result of each exponentiation separately.

3.4.2 Case $p \equiv 2^s + 1 \pmod{2^{s+1}}$

So far we have considered only one example of the characteristic p, namely $p \equiv 3 \pmod 4$. To get a broader view on the computational complexity of multiple point compression let us consider one more example.

Let $p \equiv 2^s + 1 \pmod{2^{s+1}}$ for a small positive integer s. In this case square roots can be computed using the algorithm proposed by Koo et al. [22]. The algorithm is useful for small values of s such as $s = 2, 3, 4$ since the number of branches increases exponentially as s grows. In order to compute x such that $x^2 = a$ for a given square $a \in \mathbb{F}_p$ the algorithm requires one of the following exponentiations $a^{\frac{p-5}{8}}$, $a^{\frac{p-9}{16}}$ or $a^{\frac{p-17}{32}}$ depending on whether $s = 2, 3$ or 4. This means that the square root calculation is about as expensive for $p \equiv 2^s + 1 \pmod{2^{s+1}}$ and for $p \equiv 3 \pmod 4$ for any fixed bit length. Thus the comparison tables given in the previous section are applicable for $p \equiv 2^s + 1 \pmod{2^{s+1}}$ as well.

3.4.3 Quadratic extension fields

In [1], the authors proposed two efficient algorithms for computing square roots over even field extensions of the form $\mathbb{F}_{q^2}$, with $q = p^n$, p an odd prime and $n \geq 1$. The proposed two algorithms address the case when $q \equiv 1 \pmod 4$ and $q \equiv 3 \pmod 4$, respectively, and have an associate computational cost roughly equivalent to one exponentiation in $\mathbb{F}_{q^2}$. In the case when $q \equiv 3 \pmod 4$, the authors showed that the complex method of [14] is the most efficient algorithm for square root computation. For instance, when $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ and $q = p$, the square root computation using the complex method requires $1I_p + 1149M_p + 6Mc_p$, where I_p, M_p and Mc_p denote the inversion, multiplication and multiplication by constant over the underlying $\mathbb{F}_p$. Hence, decompressing two points costs $2I_p + 2298M_p + 12Mc_p$. Using the two-point decompression technique in Sect. 2.1, one only needs $I_p + 14M_p + 7Mc_p$, given that $M_{p^2} = 3M_p + Mc_p$, $S_{p^2} = 2M_p + 2Mc_p$ and $I_{p^2} = I_p + 4M_p + Mc_p$. Using the same complexity estimation as in Sect. 3.4.1, one can find that the two-point compression technique requires 87.5 % less time when compared to the traditional approach. A similar analysis is also applicable to the case when $q \equiv 1 \pmod 4$ and we omit the details here.

3.4.4 Small characteristic fields

Elliptic curves over binary fields have the shape $y^2 + xy = f(x)$ or $y^2 + ay = f(x)$ for some cubic polynomial f and constant a. In this case, the trivial compression method stores x and one bit. To decompress we have to solve a quadratic Artin–Schreier equation $y^2 + by = c$ instead of taking a square root.

For elliptic curves defined over $\mathbb{F}_{2^m}$ (m is odd), solving a quadratic equation is highly efficient when compared to other finite fields. As shown experimentally in [18], the time complexity is approximately 2/3 the time of a field multiplication. Even in this extreme case, it is not difficult to demonstrate that the two-point decompression technique (which requires $I + 3M + S$ [20]) is roughly 50 % faster than the traditional method (which requires $2 \cdot \left(I + \frac{5}{3}M + S\right)$) for decompressing two points on elliptic curves defined over binary fields, provided that $I \geq 10M$ and $S = 0.1M$ [6].

For elliptic curves defined over $\mathbb{F}_{3^m}$ (m is odd), the best algorithm for square root extraction takes $O(m^2 \log m)$ steps [7]. If $m = 2k+1$ for some k, it is easily verified that the square root computation requires $\left(\lfloor \log_2 k \rfloor + \omega(k) + 1\right) \cdot M$ in this case, where $\omega(k)$ is the Hamming weight of the binary representation of k. In the case of $m = 239$, extracting a square root over $\mathbb{F}_{3^{239}}$ requires $13M$. While the traditional method requires $26M$ (i.e., two square root computations) for decompressing two points, the proposed two-point decompression technique only requires $1I + 2M + S$, which results in about 50 % performance improvement. Here we assume that $I = 10M$, $S = M$ and a Type-II normal basis is used, as illustrated in [2]. If $I = 30M$ the computational complexity of the two-point decompression is roughly the same as that of the traditional method.

3.4.5 Application to resource-constrained environments

When applying multiple point compression technique to resource-constrained environments such as wireless sensor networks, the performance improvement over ordinary point decompression method is further amplified due to the low-power processor architecture design. For instance, when implementing the 192-bit NIST curve on an 8-bit AVR microcontroller, the multiplication, squaring and inversion operations take $4,042$, $2,658$ and $280,829$ clock cycles, respectively [24], which gives us $I \approx 70M$, $S \approx 0.65M$ and $R = 127M + 189S \approx 250M$. Similar to the above estimations, we can obtain that the two-, three-, four- and five-point decompression algorithms requires 85.3, 88.5, 86.8 and 84.2 % less time than the traditional approach, respectively, which further demonstrates the great advantages of using multiple point compression technique on embedded devices.

4 A unified treatment of n-point compression

Considering the $\mathbb{F}$-rational generic points $P_i = (x_i, y_i)$, $(i = 1, \ldots, n)$, lying on the curve $y^2 = f(x)$, we now describe a fully LA method to compress-decompress these points using only $n + 1$ field values. When referring to a "constant number of operations" we mean that this number does not depend on the field size, but only on n and the polynomial f. Our method shows that *asymptotically*, one can achieve a very fast compression–decompression algorithm by storing only one extra value, compared to the trivial algorithm which stores only the x_i's. However, in practice, ad hoc methods such as those presented in the previous sections will be more efficient. Also, there are exceptional cases which in applications would not arise but can be dealt theoretically by storing a fixed number of extra bits, as in [20], hence our focus on an algorithm for generic points.[2] Additionally, with cryptographic applications in mind, we assume that $n < \operatorname{char} \mathbb{F}$ to avoid dividing by zero.

To compress n generic points, simply store the values

$$\left(x_1, \ldots, x_n, a_1 = y_1 + \cdots + y_n\right). \tag{20}$$

[2] Meaning that the 2^n sums $\pm y_1 \pm \cdots \pm y_n$ are all distinct.

To decompress, let $a_1, \ldots, a_n$ as in Sect. 3 be the elementary symmetric polynomials in the y_i's. Then, by Newton's identities,

$$\begin{aligned} y_1^2 + \cdots + y_n^2 &= a_1^2 - 2a_2, \\ y_1^4 + \cdots + y_n^4 &= a_1^4 - 4a_1^2a_2 + 2a_2^2 + 4a_1a_3 - 4a_4, \\ y_1^6 + \cdots + y_n^6 &= a_1^6 - 6a_1^4a_2 + 9a_1^2a_2^2 + 6a_1^3a_3 - 2a_2^3 - 12a_1a_2a_3 \\ &\quad - 6a_1^2a_4 + 3a_3^2 + 6a_2a_4 + 6a_1a_5 - 6a_6, \end{aligned} \tag{21}$$

and so on for $p_{2k} = y_1^{2k} + \cdots + y_n^{2k}$ for $k = 1, \ldots, n$. The left-hand sides are computable in a constant number of operations. The right-hand sides are polynomials $f_{2k}(a_1, \ldots, a_n)$, so the previous system (21) can be rewritten as $p_{2k} = f_{2k}(a_1, \ldots, a_n)$ for $k = 1, \ldots, n$. One can solve it in $a_1, \ldots, a_n$ by considering the ideal $\mathcal{I}$ of $\mathbb{F}(p_2, \ldots, p_{2n})[a_1, \ldots, a_n]$ generated by the $p_{2k} - f_{2k}(a_1, \ldots, a_n)$ for $k = 1, \ldots, n$ and computing its reduced Gröbner basis with respect to the lexicographic ordering of the variables with $a_1 < a_2 < \cdots < a_n$. In fact, the Newton identities applied to the symmetric polynomials in the y_i^2's show that the knowledge of p_{2k} for $k = 1, \ldots, n$ determines $y_1^2, \ldots, y_n^2$ uniquely up to permutation. By our assumption, this means that given a_1 there are either none or unique values (up to permutation) of $y_1, \ldots y_n$ for which their symmetric polynomials $(a_1, \ldots a_n)$ belong to the variety V associated to $\mathcal{I}$. Therefore $\#V = 2^n$ and the points of V are separated by a_1 (they have all distinct first coordinate). Let's note that in this reasoning we have kept in mind specializations of the p_{2k} to particular values in $\mathbb{F}$ corresponding to generic $y_1, \ldots y_n$. In fact, what this shows is that as we work with undetermined coefficients, i.e. with $\mathcal{I} \subset \mathbb{F}(p_2, \ldots, p_{2n})[a_1, \ldots, a_n]$ we have a fortiori $\#V = 2^n$, where $V \subset \overline{\left(\mathbb{F}(p_2, \ldots, p_{2n})\right)}^n$, as well as separation by the first coordinate a_1.

Moreover $\mathcal{I}$ is a radical ideal: consider again the symmetric polynomials $e_1 = y_1^2 + \cdots + y_n^2$, …, $e_n = y_1^2 \ldots y_n^2$ in the y_i^2's. We write the identity

$$\left(x^2 - y_1^2\right) \cdots \left(x^2 - y_n^2\right) = (x - y_1) \cdots (x - y_n)(x + y_1) \cdots (x + y_n)$$

as

$$\begin{aligned} &x^{2n} - e_1x^{2n-2} + e_2x^{2n-4} + \cdots + (-1)^n e_n \\ &\quad = \left(x^n - a_1x^{n-1} + a_2x^{n-2} + \cdots + (-1)^n a_n\right) \\ &\qquad \times \left(x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_n\right) \end{aligned}$$

This gives rise to the following system of n equations

$$\begin{aligned} e_1 &= -2a_2 + a_1^2 \ , \\ e_2 &= 2a_4 - 2a_1a_3 + a_2^2 \ , \\ e_3 &= -2a_6 + 2a_1a_5 - 2a_2a_4 + a_3^2 \ , \\ &\text{etc.} \end{aligned} \tag{22}$$

Newton's identities imply that $\mathbb{F}[p_2, \ldots, p_{2n}] = \mathbb{F}[e_1, \ldots, e_n]$ since we are supposing that $\operatorname{char} \mathbb{F} > n$ and hence the two systems (21) and (22) are equivalent in the sense that the ideals they define are equal in $\mathbb{F}(p_2, \ldots, p_{2n})[a_1, \ldots, a_n] = \mathbb{F}(e_1, \ldots, e_n)[a_1, \ldots, a_n]$. Looking at $\mathcal{I}$ as described by (22), we apply Bézout's Theorem for hypersurfaces [16, Ch.12.3]: since we already know that the n hyperquadrics (22) intersect in 2^n points (the maximum allowed without common components), the multiplicity of intersection at each point is one

and therefore by [13, Ch.4, Corollary 2.6] we deduce that $\mathcal{I}$ is radical (we don't need the hypothesis that the field k be algebraically closed if we have already found $V(\mathcal{I})$).

The Shape Lemma [12,30] says that under the preceding conditions, the reduced Gröbner basis will look as follows.

$$[g_1(a_1), a_2 + g_2(a_1), \ldots, a_n + g_n(a_1)] \ ,$$

where $g_k \in \mathbb{F}(p_2, \ldots, p_{2n})[a_1]$ with $\deg g_1 = 2^n$ and $\deg g_k < 2^n$ for $k = 2, \ldots, n$. This Gröbner basis can be precomputed (it depends only on n) and from its form, given a_1, the other a_k's for $k = 2, \ldots, n$ can be recovered in a constant number of operations. See Appendix 1 for the expressions of g_k when $n = 4$.

Once we know the symmetric polynomials, we can compute the right-hand sides of

$$\begin{aligned}
&y_1 + \cdots + y_n = a_1,\\
&y_1^3 + \cdots + y_n^3 = a_1^3 - 3a_1a_2 + 3a_3,\\
&y_1^5 + \cdots + y_n^5 = a_1^5 - 5a_1^3a_2 + 5a_1a_2^2 + 5a_1^2a_3 - 5a_2a_3 - 5a_1a_4 + 5a_5,\\
&\qquad\vdots
\end{aligned}$$

up to $y_1^{2n-1} + \cdots + y_n^{2n-1}$. Since

$$y_1^{2k-1} + \cdots + y_n^{2k-1} = y_1 \cdot y_1^{2k-2} + \cdots + y_n \cdot y_n^{2k-2}$$

we will get n linear equations in the n variables $y_1, \ldots, y_n$, whose matrix is the Vandermonde

$$\begin{pmatrix} 1 & \cdots & 1 \\ y_1^2 & \cdots & y_n^2 \\ \vdots & & \vdots \\ y_1^{2n-2} & \cdots & y_n^{2n-2} \end{pmatrix}$$

which will be invertible since $y_i^2 \neq y_j^2$ for $1 \leq i < j \leq n$ (otherwise not all 2^n sums $\pm y_1 \pm \cdots \pm y_n$ would be distinct). We can solve this system by linear algebra in a constant number of operations, thereby concluding the LA decompression. Thus we have proved the following theorem.

Theorem 2 *Given a curve $y^2 = f(x)$ over a field $\mathbb{F}$ of odd characteristic greater than n and $\mathbb{F}$-rational generic points $P_i = (x_i, y_i), i = 1, \ldots, n$ on it, we can fully LA compress and decompress them by storing $n + 1$ field elements.*

5 Conclusion

In this paper, we presented novel LA based compression techniques for multiple points on elliptic curves. Four- and five-point compression schemes are first described which extend Khabbazian et al. [20]. We then extend this analysis to deal with n points by generically storing $n + 1$ field elements. Using our new algorithms, the compression process requires almost no computational effort and the decompression does not involve solving any quadratic or higher degree equations over finite fields. Hence, the proposed techniques can be employed to reduce the required bandwidth/memory when single point compression is computationally expensive. It is possible to generalise this method to deal with divisors on hyperelliptic curves represented in Mumford form. One aspect that would be interesting to explore is

the possibility to design a fully LA algorithm where compression and decompression take comparable amount of time, or even where decompression is fast but compression slower, in contrast with the present algorithm where compression is immediate and decompression is much slower.

Acknowledgements We would like to thank Changbo Chen for the discussion about Gröbner bases and specifically about pointing us to the Shape Lemma. The project is financially supported by the grant of the Corporate Fund "Fund of Social Development" to Adilet Otemissov and Francesco Sica.

Appendix 1: Expression of the polynomials g_i when $n = 4$

$$
\begin{aligned}
g_1(a_1) = {} & a_1^{16} + (-8p_2)\,a_1^{14} + \left(20p_2^2 + 8p_4\right)a_1^{12} + \left(-\frac{88}{3}p_2^3 + 16p_2p_4 - \frac{128}{3}p_6\right)a_1^{10} \\
& + \left(-\frac{94}{3}p_2^4 + 360p_2^2p_4 - 248p_4^2 - \frac{1664}{3}p_2p_6 + 544p_8\right)a_1^8 \\
& + \left(72p_2^5 - \frac{1696}{3}p_2^3p_4 + 480p_2p_4^2 + 768p_2^2p_6 - \frac{512}{3}p_4p_6 - 640p_2p_8\right)a_1^6 \\
& + \left(-\frac{428}{9}p_2^6 + \frac{1192}{3}p_2^4p_4 - 624p_2^2p_4^2 - \frac{4352}{9}p_2^3p_6 + 480p_4^3\right. \\
& \quad \left. + \frac{512}{3}p_2p_4p_6 + 576p_2^2p_8 + \frac{4096}{9}p_6^2 - 896p_4p_8\right)a_1^4 \\
& + \left(-\frac{56}{9}p_2^7 + \frac{304}{3}p_2^5p_4 - 480p_2^3p_4^2 - \frac{896}{9}p_2^4p_6 + 320p_2p_4^3 + \frac{3584}{3}p_2^2p_4p_6\right. \\
& \quad \left. - \frac{128}{3}p_2^3p_8 - 512p_4^2p_6 - \frac{8192}{9}p_2p_6^2 - 256p_2p_4p_8 + \frac{2048}{3}p_6p_8\right)a_1^2 \\
& + \frac{25}{9}p_2^8 - 40p_2^6p_4 + \frac{472}{3}p_2^4p_4^2 + \frac{640}{9}p_2^5p_6 - 96p_2^2p_4^3 \\
& - 512p_2^3p_4p_6 - \frac{160}{3}p_2^4p_8 \\
& + 16p_4^4 + \frac{512}{3}p_2p_4^2p_6 + \frac{4096}{9}p_2^2p_6^2 + 384p_2^2p_4p_8 - 128p_4^2p_8 \\
& - \frac{2048}{3}p_2p_6p_8 + 256p_8^2\ , \\
g_2(a_1) = {} & \left(-\frac{1}{2}\right)a_1^2 + \frac{1}{2}p_2.
\end{aligned}
$$

Next, $g_3(a_1)$ equals

$$
\begin{aligned}
& (3141p_2^6 - 29916p_2^4p_4 + 73575p_2^2p_4^2 + 27288p_2^3p_6 - 22950p_4^3 \\
& - 107568p_2p_4p_6 - 9558p_2^2p_8 + 41472p_6^2 + 24300p_4p_8)a_1^{15} \\
& + (-24228p_2^7 + 229068p_2^5p_4 - 559224p_2^3p_4^2 - 203544p_2^4p_6 \\
& + 174528p_2p_4^3 + 788832p_2^2p_4p_6 \\
& + 67824p_2^3p_8 + 7776p_4^2p_6 - 290304p_2p_6^2 - 158112p_2p_4p_8 - 31104p_6p_8)a_1^{13}
\end{aligned}
$$

$$
\begin{aligned}
&+(56205p_2^8-498654p_2^6p_4+1024839p_2^4p_4^2+438408p_2^5p_6+177660p_2^2p_4^3\\
&-1425312p_2^3p_4p_6\\
&-130086p_2^4p_8-178092p_4^4-885600p_2p_4^2p_6+539136p_2^2p_6^2\\
&+168696p_2^2p_4p_8+331776p_4p_6^2+166536p_4^2p_8+186624p_2p_6p_8+23328p_8^2)a_1^{11}\\
&+(-78726p_2^9+789120p_2^7p_4-2350998p_2^5p_4^2-722856p_2^6p_6\\
&+2106360p_2^3p_4^3+4069584p_2^4p_4p_6\\
&+169452p_2^5p_8-486216p_2p_4^4-5601312p_2^2p_4^2p_6-1869312p_2^3p_6^2-589680p_2^3p_4p_8\\
&+1046592p_4^3p_6\\
&+5612544p_2p_4p_6^2+907632p_2p_4^2p_8+252288p_2^2p_6p_8-1769472p_6^3\\
&-1306368p_4p_6p_8-171072p_2p_8^2)a_1^9\\
&+(-113973p_2^{10}+2247624p_2^8p_4-14414355p_2^6p_4^2-2873976p_2^7p_6+35073162p_2^4p_4^3\\
&+31435920p_2^5p_4p_6+2121606p_2^6p_8-26740476p_2^2p_4^4-88583328p_2^3p_4^2p_6\\
&-17567232p_2^4p_6^2\\
&-20887092p_2^4p_4p_8+5739336p_4^5+40201920p_2p_4^3p_6\\
&+78474240p_2^2p_4p_6^2+51563304p_2^2p_4^2p_8\\
&+20431872p_2^3p_6p_8-10616832p_4^2p_6^2-24772608p_2p_6^3\\
&-18752688p_4^3p_8-74428416p_2p_4p_6p_8\\
&-4854816p_2^2p_8^2+23887872p_6^2p_8+13421376p_4p_8^2)a_1^7\\
&+(183552p_2^{11}-3089004p_2^9p_4+18070524p_2^7p_4^2+3134904p_2^8p_6-43127352p_2^5p_4^3\\
&-31349184p_2^6p_4p_6-1758456p_2^7p_8+37337904p_2^3p_4^4\\
&+86126976p_2^4p_4^2p_6+13088256p_2^5p_6^2\\
&+15252912p_2^5p_4p_8-8725536p_2p_4^5-54885888p_2^2p_4^3p_6\\
&-52205568p_2^3p_4p_6^2-32713632p_2^3p_4^2p_8\\
&-9452160p_2^4p_6p_8+1798272p_4^4p_6+22321152p_2p_4^2p_6^2\\
&+7077888p_2^2p_6^3+12156480p_2p_4^3p_8\\
&+15422976p_2^2p_4p_6p_8+1019520p_2^3p_8^2-7077888p_4p_6^3+8805888p_4^2p_6p_8\\
&+15925248p_2p_6^2p_8+4582656p_2p_4p_8^2-17915904p_6p_8^2)a_1^5\\
&+(-119357p_2^{12}+2165658p_2^{10}p_4-14673531p_2^8p_4^2\\
&-2088552p_2^9p_6+46760208p_2^6p_4^3\\
&+23773056p_2^7p_4p_6+1897398p_2^8p_8-73942056p_2^4p_4^4
\end{aligned}
$$

$$
\begin{aligned}
&- 86667456p_2^5p_4^2p_6 - 9014784p_2^6p_6^2\\
&- 22437504p_2^6p_4p_8 + 57054240p_2^2p_4^5 + 117450240p_2^3p_4^3p_6\\
&+ 43920384p_2^4p_4p_6^2 + 87558768p_2^4p_4^2p_8\\
&+ 17627904p_2^5p_6p_8 - 12932784p_4^6 - 68144256p_2p_4^4p_6\\
&- 20305920p_2^2p_4^2p_6^2 - 3506176p_2^3p_6^3\\
&- 124350336p_2^2p_4^3p_8 - 111430656p_2^3p_4p_6p_8 - 6348672p_2^4p_8^2\\
&+ 8687616p_4^3p_6^2 - 46006272p_2p_4p_6^3\\
&+ 45537120p_4^4p_8 + 143926272p_2p_4^2p_6p_8 + 36790272p_2^2p_6^2p_8\\
&+ 41478912p_2^2p_4p_8^2 + 18874368p_6^4\\
&- 23003136p_4p_6^2p_8 - 48169728p_4^2p_8^2 - 35665920p_2p_6p_8^2 + 15303168p_8^3)a_1^3\\
&\quad + (5186p_2^{13} - 10704p_2^{11}p_4 - 936990p_2^9p_4^2 + 105288p_2^{10}p_6\\
&\quad + 8434128p_2^7p_4^3 + 1229136p_2^8p_4p_6\\
&\quad - 370452p_2^9p_8 - 26146800p_2^5p_4^4 - 23498496p_2^6p_4^2p_6 - 907776p_2^7p_6^2\\
&\quad + 3779424p_2^7p_4p_8 + 28215360p_2^3p_4^5\\
&\quad + 92255232p_2^4p_4^3p_6 + 26336256p_2^5p_4p_6^2 - 9767520p_2^5p_4^2p_8\\
&\quad - 3057024p_2^6p_6p_8 - 7140960p_2p_4^6\\
&\quad - 90170496p_2^2p_4^4p_6 - 132544512p_2^3p_4^2p_6^2 - 8880128p_2^4p_6^3\\
&\quad - 3844224p_2^3p_4^3p_8 + 9347328p_2^4p_4p_6p_8\\
&\quad + 2239488p_2^5p_8^2 + 13130496p_4^5p_6 + 87527424p_2p_4^3p_6^2\\
&\quad + 88473600p_2^2p_4p_6^3 + 11524032p_2p_4^4p_8\\
&\quad + 60120576p_2^2p_4^2p_6p_8 - 9338880p_2^3p_6^2p_8 - 15123456p_2^3p_4p_8^2\\
&\quad - 17694720p_4^2p_6^3 - 18874368p_2p_6^4\\
&\quad - 36615168p_4^3p_6p_8 - 85819392p_2p_4p_6^2p_8 + 2087424p_2p_4^2p_8^2\\
&\quad + 25436160p_2^2p_6p_8^2 + 14155776p_6^3p_8\\
&\quad + 28532736p_4p_6p_8^2 - 12192768p_2p_8^3)a_1
\end{aligned}
$$

divided by

$$
\begin{aligned}
&82560p_2^{12} - 1615872p_2^{10}p_4 + 11757312p_2^8p_4^2 + 1855488p_2^9p_6 - 38739456p_2^6p_4^3\\
&- 24600576p_2^7p_4p_6 - 1230336p_2^8p_8 + 55052928p_2^4p_4^4 + 105615360p_2^5p_4^2p_6\\
&+ 12435456p_2^6p_6^2 + 16137216p_2^6p_4p_8 - 26417664p_2^2p_4^5 - 150552576p_2^3p_4^3p_6\\
&- 92061696p_2^4p_4p_6^2 - 67834368p_2^4p_4^2p_8 - 16588800p_2^5p_6p_8 + 3995136p_4^6
\end{aligned}
$$

$$+41213952p_2p_4^4p_6+127401984p_2^2p_4^2p_6^2+28311552p_2^3p_6^3+90740736p_2^2p_4^3p_8$$
$$+120029184p_2^3p_4p_6p_8+6068736p_2^4p_8^2-5308416p_4^3p_6^2-28311552p_2p_4p_6^3$$
$$-24440832p_4^4p_8-132710400p_2p_4^2p_6p_8-63700992p_2^2p_6^2p_8-43960320p_2^2p_4p_8^2$$
$$+21233664p_4p_6^2p_8+38320128p_4^2p_8^2+55738368p_2p_6p_8^2-17915904p_8^3.$$

The formula for $g_4(a_1)$ is

$$(-180p_2^3+756p_2p_4-648p_6)a_1^{14}$$
$$+(1323p_2^4-5382p_2^2p_4-459p_4^2+4536p_2p_6+486p_8)a_1^{12}$$
$$+(-2682p_2^5+8424p_2^3p_4+9918p_2p_4^2-7992p_2^2p_6-5616p_4p_6-3564p_2p_8)a_1^{10}$$
$$+(3111p_2^6-13500p_2^4p_4+7389p_2^2p_4^2+15528p_2^3p_6-3978p_4^3-45072p_2p_4p_6$$
$$+7182p_2^2p_8+27648p_6^2+4212p_4p_8)a_1^8$$
$$+(8520p_2^7-106788p_2^5p_4+340092p_2^3p_4^2+139368p_2^4p_6-190872p_2p_4^3$$
$$-696480p_2^2p_4p_6-106200p_2^3p_8+176544p_4^2p_6$$
$$+387072p_2p_6^2+419472p_2p_4p_8-373248p_6p_8)a_1^6$$
$$+(-8067p_2^8+83322p_2^6p_4-209049p_2^4p_4^2-88728p_2^5p_6+11412p_2^2p_4^3$$
$$+347040p_2^3p_4p_6+26586p_2^4p_8+118116p_4^4-9696p_2p_4^2p_6-119808p_2^2p_6^2$$
$$+75528p_2^2p_4p_8+119808p_4p_6^2-382104p_4^2p_8-235008p_2p_6p_8+272160p_8^2)a_1^4$$
$$+(-818p_2^9+16344p_2^7p_4-104490p_2^5p_4^2-14312p_2^6p_6+177288p_2^3p_4^3$$
$$+244944p_2^4p_4p_6-10188p_2^5p_8+66312p_2p_4^4-618336p_2^2p_4^2p_6-180224p_2^3p_6^2$$
$$+5040p_2^3p_4p_8-115008p_4^3p_6+645120p_2p_4p_6^2-66672p_2p_4^2p_8+161280p_2^2p_6p_8$$
$$-294912p_6^3+248832p_4p_6p_8-160704p_2p_8^2)a_1^2$$
$$+3809p_2^{10}-58848p_2^8p_4+320919p_2^6p_4^2+57464p_2^7p_6-707514p_2^4p_4^3$$
$$-565968p_2^5p_4p_6-33006p_2^6p_8+532188p_2^2p_4^4+1485984p_2^3p_4^2p_6+272384p_2^4p_6^2$$
$$+312660p_2^4p_4p_8-113832p_4^5-721344p_2p_4^3p_6-1096704p_2^2p_4p_6^2-749448p_2^2p_4^2p_8$$
$$-296448p_2^3p_6p_8+165888p_4^2p_6^2+294912p_2p_6^3+331056p_4^3p_8+912384p_2p_4p_6p_8$$
$$+97632p_2^2p_8^2-221184p_6^2p_8-222912p_4p_8^2$$

divided by

$$-16512p_2^8+204288p_2^6p_4-840960p_2^4p_4^2-159744p_2^5p_6+1202688p_2^2p_4^3$$
$$+1155072p_2^3p_4p_6+87552p_2^4p_8-332928p_4^4-1658880p_2p_4^2p_6-442368p_2^2p_6^2$$
$$-635904p_2^2p_4p_8+442368p_4p_6^2+705024p_4^2p_8+663552p_2p_6p_8-373248p_8^2\ .$$

These formulas were obtained by running the following commands in SAGE (www.sagemath.org):

```
Rinit.<p2,p4,p6,p8>=QQ['p2,p4,p6,p8]; Finit=Rinit.fraction_field();

R0.<a1,a2,a3,a4>=Finit['a1,a2,a3,a4'];R=R0.change_ring(order='invlex');

r1=a1^2 - 2*a2-p2;

r2=a1^4 - 4*a1^2*a2 + 2*a2^2 + 4*a1*a3 - 4*a4-p4;

r3=a1^6 - 6*a1^4*a2 + 9*a1^2*a2^2 + 6*a1^3*a3
- 2*a2^3 - 12*a1*a2*a3 -6*a1^2*a4 + 3*a3^2 + 6*a2*a4-p6;

r4= a1^8 - 8*a1^6*a2 + 20*a1^4*a2^2 + 8*a1^5*a3 - 16*a1^2*a2^3 -32*a1^3*a2*a3
- 8*a1^4*a4 + 2*a2^4+24*a1*a2^2*a3 + 12*a1^2*a3^2 +24*a1^2*a2*a4 - 8*a2*a3^2
- 8*a2^2*a4 - 16*a1*a3*a4 +4*a4^2 - p8;

I=(r1,r2,r3,r4)*R;

B=I.groebner_basis()
```

Appendix 2: Computational complexity of the decompression for $n = 4$

All points are decompressed

The calculation of the complexity is based on the principles described in [23], which are as follows.

- Only the square and multiply operations are counted; the computational cost of the addition and subtraction is neglected.
- Multiplication by a constant is neglected as well.

It most cases inversion is computationally costly as compared to multiplication and square operations. For that reason, we will minimize the number of the inversions.

Denote

$$\begin{aligned} B &= a_1\left(A^2 + a_4^2 - a_1^2 b_3\right), \\ C &= 2a_1\left(a_1^2 a_2 - A\right), \\ D &= 2a_1^2 a_2 A - A^2 + a_4^2 - a_1^2 b_3, \\ E &= 2a_1^2\left(a_1^2 a_2 - A\right). \end{aligned}$$

Then Eq. (10) yields

$$y_i = \frac{B + C\left(a_2 y_i^2 + y_i^4\right)}{D + E y_i^2}. \tag{23}$$

The computation of y_1, y_2, y_3, y_4 separately using the equation above ends up with 4 inversions. For the sake of efficiency, it makes sense to compute y_i's in the following way:

$$y_1 = \frac{\left(B + C\left(a_2 y_1^2 + y_1^4\right)\right)\left(D + E y_2^2\right)\left(D + E y_3^2\right)\left(D + E y_4^2\right)}{\left(D + E y_1^2\right)\left(D + E y_2^2\right)\left(D + E y_3^2\right)\left(D + E y_4^2\right)},$$

$$y_2 = \frac{\left(B + C\left(a_2 y_2^2 + y_2^4\right)\right)\left(D + E y_1^2\right)\left(D + E y_3^2\right)\left(D + E y_4^2\right)}{\left(D + E y_1^2\right)\left(D + E y_2^2\right)\left(D + E y_3^2\right)\left(D + E y_4^2\right)},$$

Table 1 Computational complexity of the decompression algorithm

Expression	Cost	Comments
$y_1^2, y_2^2, y_3^2, y_4^2$	$4S + 4M$	This comes from $y^2 = x(x^2 + a) + b$
$b_2 = y_1^2y_2^2 + y_1^2y_3^2 + y_1^2y_4^2 + y_2^2y_3^2 + y_2^2y_4^2 + y_3^2y_4^2$	$6M$	Store $y_1^2y_2^2$, $y_1^2y_3^2$, $y_1^2y_4^2$, $y_2^2y_3^2$
$b_3 = y_1^2y_2^2y_3^2 + y_1^2y_2^2y_4^2 + y_1^2y_3^2y_4^2 + y_2^2y_3^2y_4^2$	$4M$	Store $y_1^2y_2^2y_3^2$
$a_4^2 = y_1^2y_2^2y_3^2y_4^2$	$1M$	Stored value of $y_1^2y_2^2y_3^2$ is multiplied by y_4^2
$y_1^4, y_2^4, y_3^4, y_4^4$	$4S$	y_i^2's are each squared
a_1^2	1S	a_1 is squared
a_2	0	a_2 is computed only through addition/subtraction operations and a multiplication by a constant
a_2^2	$1S$	a_2 is squared
A	0	Similarly, this does not require any of M, S or I
A^2	$1S$	A is squared
$B = a_1(A^2 + a_4^2 - a_1^2b_3)$	$2M$	
$C = 2a_1(a_1^2a_2 - A)$	$2M$	
$D = 2a_1^2a_2A - A^2 + a_4^2 - a_1^2b_3$	$3M$	
$E = 2a_1^2(a_1^2a_2 - A)$	$2M$	
$D + Ey_1^2$	$1M$	
$D + Ey_2^2$	$1M$	
$D + Ey_3^2$	$1M$	
$D + Ey_4^2$	$1M$	
$(D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)(D + Ey_4^2)$	$3M$	
$\left((D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)(D + Ey_4^2)\right)^{-1}$	$1I$	
$B + C(a_2y_1^2 + y_1^4)$	$2M$	
$B + C(a_2y_2^2 + y_2^4)$	$2M$	
$B + C(a_2y_3^2 + y_3^4)$	$2M$	
$B + C(a_2y_4^2 + y_4^4)$	$2M$	
Final computation of y_1, y_2, y_3, y_4	$4M \cdot 4 = 16M$	

$$y_3 = \frac{\left(B + C(a_2y_3^2 + y_3^4)\right)(D + Ey_1^2)(D + Ey_2^2)(D + Ey_4^2)}{(D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)(D + Ey_4^2)},$$

$$y_4 = \frac{\left(B + C(a_2y_4^2 + y_4^4)\right)(D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)}{(D + Ey_1^2)(D + Ey_2^2)(D + Ey_3^2)(D + Ey_4^2)}.$$

Here all y_i's are converted to the common denominator, which means that only one inversion is needed.

Table 1 shows how the computational complexity is calculated.

Overall, the decompression algorithm require $55M + 11S + 1I$.

Table 2 Computational complexity of the preparation step

Expression	Cost	Comments
$y_1^2, y_2^2, y_3^2, y_4^2, y_5^2$	$5S + 5M$	Comes from $y^2 = x^3 + ax + b$
$b_2 = y_1^2y_2^2 + y_1^2y_3^2 + \cdots + y_4^2y_5^2$	$10M$	Store only $y_1^2y_2^2, y_1^2y_3^2, y_2^2y_3^2, y_4^2y_5^2$
$b_3 = y_1^2y_2^2y_3^2 + \cdots + y_3^2y_4^2y_5^2$	$10M$	Store only $y_1^2y_2^2y_3^2, y_1^2y_4^2y_5^2, y_2^2y_3^2y_4^2$
$b_4 = y_1^2y_2^2y_3^2y_4^2 + y_1^2y_2^2y_3^2y_5^2 + \cdots + y_2^2y_3^2y_4^2y_5^2$	$5M$	
$b_5 = a_5^2 = y_1^2y_2^2y_3^2y_4^2y_5^2$	M	
a_1^2	S	
a_2	0	Only addition is required
a_2^2	S	
A	0	Does not require any M, S, I
$C = 2a_1\left(a_1^2a_2 - A\right)$	$2M$	
a_1^4	S	
$F = 2a_1^4 + 2A$	0	
A^2	S	
$G = \left(A^2 + b_4 - b_3a_1^2\right) \cdot a_1$	$2M$	
C^2	S	
First component of H: $a_1\left(2FG + 4Cb_4\right)$	$3M$	
Second component of H: $A\left(-2C^2 - 8a_5^2\right)$	M	Recall that a_5^2, C^2 have been computed
Third component of H: $-2GC + 4Fa_5^2$	$2M$	
H	0	Addition of the three components
1st component of J: $a_1\left(-F^2a_5^2 - G^2 + C^2b_4 + 4b_4a_5^2\right)$	$4M + 2S$	Recall that a_5^2, C^2 have been computed
Second component of J: $C\left(-8Aa_5^2 + 2Fa_5^2\right)$	$3M$	
Third component of J: $-4Ga_5^2$	M	
J	0	Addition of the three components
$K = CH - 2J$	M	
$L = FJ + GH$	$2M$	

Only one point is decompressed

The complexity of the decompression can be reduced when one has to extract only one $y_i, i \in \{1, 2, 3, 4\}$, and not all of them. Note that in many practical situations only one of the y-coordinates has to be extracted for the archive.

Without loss of generality, suppose we have to recover y_1. We will compute y_1 using Eq. (23).

Table 3 Computational complexity of the final computation: all y_i's are retrieved

Expression	Cost	Comments
$Ka_1^2y_i^2, i = 1, 2, 3, 4, 5$	$2M \cdot 5 = 10M$	a_1^2, y_i^2 have already been computed
Hy_i^2	$M \cdot 5 = 5M$	
$AK + L$	M	
JKa_1	$2M$	
$M_i = JKa_1 + Hy_i^2(AK + L + Ka_1^2y_i^2)$, $i = 1, 2, 3, 4, 5$	$M \cdot 5 = 5M$	Numerator of y_i in (18)
$L + y_i^2K(a_2 + y_i^2), i = 1, 2, 3, 4, 5$	$2M \cdot 5 = 10M$	
Ha_1	M	
$Ha_1\left(L + y_i^2K(a_2 + y_i^2)\right), i = 1, 2, 3, 4, 5$	$M \cdot 5 = 5M$	Denominator of y_i in (18)
$N = Ha_1\left(L + y_1^2K(a_2 + y_1^2)\right) \cdots \left(L + y_5^2K(a_2 + y_5^2)\right)$	$4M$	Common denominator
N^{-1}	I	Inversion of the common denominator
Final computation of $y_i, i = 1, 2, 3, 4, 5$	$5M \cdot 5 = 25M$	See (19)

Table 4 Computational complexity of the final computation: only y_1 is retrieved

Expression	Cost	Comments
$Ka_1^2y_1^2$	$2M$	a_1^2, y_1^2 have already been computed
Hy_1^2	M	
$AK + L$	M	
JKa_1	$2M$	
$JKa_1 + Hy_1^2(AK + L + ka_1^2y_1^2)$	M	Numerator of y_1 in (18)
$L + y_1^2K(a_2 + y_1^2)$	$2M$	
Ha_1	M	
$Ha_1(L + y_1^2K(a_2 + y_1^2))$	M	Denominator of y_1 in (18)
$Ha_1(L + y_1^2K(a_2 + y_1^2))^{-1}$	I	
Final computation of y_1	M	Product of the numerator and the inverse of the denominator in (18)

Then in Table 1 we can avoid the computation of the following values: $D+Ey_2^2$, $D+Ey_3^2$, $D+Ey_4^2$, $\left(D+Ey_1^2\right)\left(D+Ey_2^2\right)\left(D+Ey_3^2\right)\left(D+Ey_4^2\right)$, $B+C\left(a_2y_2^2+y_2^4\right)$, $B+C\left(a_2y_3^2+y_2^4\right)$, $B + C\left(a_2y_4^2 + y_2^4\right)$. This saves us $1M + 1M + 1M + 3M + 2M + 2M + 2M = 12M$.

Instead of inverting $\left(D+Ey_1^2\right)\left(D+Ey_2^2\right)\left(D+Ey_3^2\right)\left(D+Ey_4^2\right)$ we will invert $\left(D+Ey_1^2\right)$ so we still need $1I$.

In the end we needed extra $16M$ to compute y_1, y_2, y_3, y_4 (see the very end of Sect. 1). And now we will need $1M$ instead: this is the multiplication $\left(B+C\left(a_2y_1^2+y_1^4\right)\right)\cdot\left(D+Ey_1^2\right)^{-1}$. This saves us $15M$.

In total we save $12M + 15M = 27M$ and the overall complexity becomes $(55M + 11S + 1I) - 27M = 28M + 11S + 1I$.

Furthermore, in the case when the decompression algorithm may output the projective coordinates of the point (which is acceptable e.g. when the decompressed point is used for the point addition or point doubling) we can avoid the inversion.

Appendix 3: Computational complexity of the decompression for $n = 5$

Preparation step

Table 2 shows the cost of computing $y_i^2, b_i, i = 1, 2, 3, 4, 5$ as well as C, F, G, H, K, L.

Final computation of y_i

Table 3 demonstrates the cost of the computation of y_i, $i = 1, 2, 3, 4, 5$ in the case when the goal is to retrieve the y-coordinates of all five points. Table 4 shows the cost of the computational complexity can be reduced significantly when the goal is to retrieve only one $y_i, i \in \{1, 2, 3, 4, 5\}$.

The overall computational complexity of the decompression algorithm is $120M + 12S + I$ in the case when the goal is to retrieve the y-coordinates of all five points. If only one point has to be retrieved the complexity reduces to $64M + 12S + I$. Furthermore, when the decompression algorithm may output the projective coordinates we can avoid the inversion.

References

1. Adj G., Rodríguez-Henríquez F.: Square root computation over even extension fields. IEEE Trans. Comput. **63**(11), 2829–2841 (2014).
2. Ahmadi O., Hankerson D., Menezes A.: Software implementation of arithmetic in $\mathbb{F}_{3^m}$. In: Carlet C., Sunar B. (eds.) Proceedings of WAIFI 2007. Lecture Notes in Computer Science, vol. 4876, pp. 85–102. Springer, Berlin (2007).
3. ANSI X9.62 Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA) (2005).
4. ANSI X9.63. Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography (2001).
5. Avanzi R.M., Cohen H., Doche D., Frey G., Lange T., Nguyen K., Vercauteren F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. Chapman & Hall/CRC, Boca Raton (2006).
6. Avanzi R.M.: Another look at square roots (and other less common operations) in fields of even characteristic. In: Adams C., Miri A., Wiener M. (eds.) Proceedings of SAC 2007. Lecture Notes in Computer Science, vol. 4876, pp. 138–154. Springer, Berlin (2007).
7. Barretol P.S.L.M., Kim H.Y., Lynn B., Scott M.: Efficient algorithms for pairing-based cryptosystems. In: Yung M. (ed.) Proceedings of CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 354–369. Springer, Berlin (2002).
8. Bellare M., Namprempre C., Neven G.: Security proofs for identity-based identification and signature schemes. In: Cachin C., Camenisch J. (eds.) Proceedings of EUROCRYPT 2004. Lecture Notes in Computer Science, vol. 3027, pp. 268–286. Springer, Berlin (2004).
9. Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., Moeller, B.: Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS). IETF Internet Draft, May 2006. http://tools.ietf.org/html/rfc4492.
10. Campagna, M., Zaverucha, G.: A cryptographic suite for embedded systems (suite E). IETF Internet Draft, October 2012. http://tools.ietf.org/html/draft-campagna-suitee-04.
11. Cohen H.: A Course in Computational Algebraic Number Theory, vol. 138. Graduate Texts in Mathematics. Springer, Berlin (1996).

12. Cox D.A., Little J., O'Shea D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra. Springer, Secaucus (2007).
13. Cox D.A., Little J., O'Shea D.: Using Algebraic Geometry. Springer, Secaucus (1998).
14. Devegili A.J., Scott M., Dahab R.: Implementing cryptographic pairings over Barreto-Naehrig curves. In: Takagi T., Okamoto E., Okamoto T., Okamoto T. (eds.) Proceedings of Pairing 2007. Lecture Notes in Computer Science, vol. 4575, pp. 197–207. Springer, Berlin (2007).
15. Fan X., Gong G.: Accelerating signature-based broadcast authentication for wireless sensor networks. Ad Hoc Netw. **10**(4), 723–736 (2012).
16. Fulton W.: Intersection Theory. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer, Berlin (1984).
17. Galindo D., Garcia F.D.: A Schnorr-like lightweight identity-based signature scheme. In: Preneel B. (ed.) Proceedings of AFRICACRYPT 2009. Lecture Notes in Computer Science, vol. 5580, pp. 135–148. Springer, Berlin (2009).
18. Hankerson D., Menezes A., Vanstone S.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2003).
19. IEEE Vehicular Technology Society. IEEE standard for wireless access in vehicular environments security services for applications and management messages (2013).
20. Khabbazian M., Gulliver T.A., Bhargava V.K.: Double point compression with applications to speeding up random point multiplication. IEEE Trans. Comput. **56**(3), 305–313 (2007).
21. Koblitz N.: Elliptic curve cryptosystems. J. Math. Comput. **48**, 203–209 (1987).
22. Koo N., Cho G.H., Kwon S.: Square root algorithm in$\mathbb{F}_q$ for $q \equiv 2^s + 1 \pmod{2^{s+1}}$. https://eprint.iacr.org/2013/087.pdf.
23. Lange T.: Explicit formulas database. http://hyperelliptic.org.
24. Liu Z., Seo H., Groschädl J., Kim H.: Efficient implementation of NIST-compliant elliptic curve cryptography for sensor node. In: Qing S., Zhou J., Liu D. (eds.) Proceedings of ICICS 2013. Lecture Notes in Computer Science, vol. 8233, pp. 302–317. Springer, Berlin (2013).
25. Lou W., Ren K., Yu S., Zhang Y.: Multi-user broadcast authentication in wireless sensor networks. IEEE Trans. Veh. Technol. **58**(8), 4554–4564 (2009).
26. National Highway Traffic Safety Administration. Vehicle safety communications project—task 3 final report, March 2005. http://www.its.dot.gov/research_docs/pdf/59vehicle-safety.pdf.
27. NIST SP 800-90. Recommendation for Random Number Generation Using Deterministic Random Bit Generators (2012)
28. NIST Removes Cryptography Algorithm from Random Number Generator Recommendations, April 2014. http://www.nist.gov/itl/csd/sp800-90-042114.cfm.
29. Peterson W.W., Brown D.T.: Cyclic Codes for Error Detection. Proc. IRE **49**(1), 228–235 (1961).
30. Rouillier F.: Solving zero-dimensional systems through the rational univariate representation. Appl. Algebra Eng. Commun. Comput. **9**(5), 433–461 (1999).
31. Vanstone S., Menezes A., van Oorschot P.: Handbook of Applied Cryptography. CRC Press, Boca Raton (2001).
32. Wollinger T., Pelzl J., Paar C., Saldamli G., Koç Ç.K.: Elliptic and hyperelliptic curves on embedded μp. ACM Trans. Embed. Comput. Syst. **3**(3), 509–533 (2004).
33. Yen S.-M., Laih C.-S., Lenstra A.K.: Multi-exponentiation. Comput. Digit. Tech. **141**(6), 325–326 (1994).
34. Zhu S., Liu D., Ning P., Jajodia S.: Practical broadcast authentication in sensor networks. In: Proceedings of the Second Annual International Conference on Mobile and Ubiquitous System: Networking and Services (MobiQuitous 2005), pp. 118–129 (2005).