Efficient Three-Term Simultaneous Elliptic Scalar Multiplication with Applications *

Billy Bob Brumley

Helsinki University of Technology,
Laboratory for Theoretical Computer Science,
P.O.Box 5400, FI-02015 TKK, Finland,
billy.brumley@hut.fi

Abstract. An application of n-term Joint Sparse Form to three-term simultaneous elliptic scalar multiplication is presented. This is shown to significantly improve performance in comparison to processing the scalar multiplications individually. A practical application of the results is provided using Self-Certified signatures. These results are particularly useful when compact and fast signatures are needed.

Key words: elliptic curve cryptography, self-certified identity based digital signature schemes, simultaneous elliptic scalar multiplication, n-term joint sparse form

1 Introduction

As "The Tragedy of the Commons" [9] is becoming an increasing reality on the Internet, new and more efficient protection methods are required. One vision [4] 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.

This is just one example of a security problem in which digital signatures, if implemented efficiently, can provide a solution. Digital signatures, in turn, require an effective Public Key Infrastructure (PKI) to work properly. One of the many alternatives to traditional certificate-based PKI is Self-Certified (SC) keys and signatures. SC signature schemes are attractive due to the reduction in computational and size requirements and are

* This work was supported by the project "Packet Level Authentication" funded by TEKES. Thanks to Prof. Kaisa Nyberg for suggestions, comments, and generous support. Additionally, the author gratefully acknowledges professors and co-researchers involved in the PLA project.

therefore well-suited for environments where fast and compact signature are needed, as in [4].

Improving efficiency in verification of SC signatures provides the motivation for this paper. In Section 2, some background information is provided on digital signature schemes using elliptic curves, minimal weight signed binary representations, joint signed binary representations, and SC schemes. In Section 3, an application of n-term Joint Sparse Form (JSF) [14] to three-term simultaneous elliptic scalar multiplication is presented. A practical application of these results is given involving a SC signature scheme [2]. In Section 4, estimates are provided which show that using simultaneous elliptic scalar multiplication and three-term JSF significantly improves performance compared to processing the scalar multiplications individually.

2 Background

This section provides a brief overview of background information that helps understand the results presented in this paper. Topics covered include the Nyberg-Rueppel signature scheme, elliptic scalar multiplication, minimal weight signed binary representations and joint expansions, simultaneous elliptic scalar multiplication, and SC signature schemes.

2.1 The Nyberg-Rueppel Signature Scheme

The Nyberg-Rueppel signature scheme was introduced in [11]. It is a variation of the ElGamal scheme [6]. In this paper, an implementation based on elliptic curves is considered. The steps of the signature scheme are outlined 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

$$\begin{aligned} s &\in_R \mathbb{Z}_r^* \\ W &= sG \end{aligned} \tag{1}$$

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

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

$$\begin{aligned} u &\in_R \mathbb{Z}_r^* \\ c &= [uG]_x + \mathrm{H}(m) \pmod r \\ d &= u - sc \pmod r \end{aligned} \tag{2}$$

where $[P]_x$ denotes the x-coordinate of the point P converted to an integer and H is a collision resistant hash function. This requires one elliptic scalar multiplication involving a fixed point G.

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

$$\text{H}(m) = c - [dG + cW]_x \pmod{r} \tag{3}$$

This 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.

Correctness. These computations are consistent:

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

2.2 Elliptic Scalar Multiplication

Arguably the most well-known and simple method for elliptic scalar multiplication is the *Double-and-Add Method*, which is analogous to the *Square-and-Multiply Method* for modular exponentiation. It uses the binary representation of a scalar k to compute multiples of the point P.

Algorithm 1: 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

Since the average number of non-zero digits in ℓ-bit k is $\ell/2$, Algorithm 1 is executed at the cost of

$$\frac{\ell}{2}A + \ell D \tag{4}$$

where A denotes point additions and D point doublings.

In multiplicative groups, inversions are much more expensive than multiplications. However, in groups over elliptic curves, point negation is free, hence the equivalent elliptic curve operations of point subtraction

and point addition have roughly the same cost. To subtract the point $P = (x, y)$, the point $-P = (x, -y)$ is added for elliptic curves over prime fields and the point $-P = (x, x + y)$ for binary fields. So when implementing signature schemes using elliptic curves, signed digit representations are preferred over unsigned digit representations.

The *Addition-Subtraction Method* presented as Algorithm 2 uses the *Non-Adjacent Form* (NAF) of a scalar k to compute multiples of the point P. This is the only method present in many standards [10, 1].

Algorithm 2: 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

NAF has the minimal weight among all signed binary digit representations. The average number of non-zero digits in the ℓ-bit scalar k using NAF is $\ell/3$, giving Algorithm 2 an average cost of

$$\frac{\ell}{3}A + \ell D \tag{5}$$

which is significantly less than the costs given in Equation 4.

Since the signature verification presented in Equation 3 requires two elliptic scalar multiplications, processing these separately and adding the results requires on average

$$(\frac{2\ell}{3} + 1)A + 2\ell D \tag{6}$$

2.3 Simultaneous Elliptic Scalar Multiplication and the Joint Sparse Form

In many signature verification primitives (including Equation 3), 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 as presented above is to calculate each term separately, then combine the result. However, calculations of the above form have specialized methods known as *simultaneous* elliptic scalar multiplication. Using a modification of *Shamir's Trick* [6] to process $kP + lQ$ in parallel can reduce the number of operations needed. The idea is that the values for the individual terms are not needed, only their sum. See Figure 1 for a small example.

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

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

Algorithm 3 modified from [8] illustrates this method. Note that if k_i, l_i are both zero, no point addition takes place (the point at infinity, which is the neutral element, is added). Improvements known as *window* methods (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.

In Equation 5, the cost of one elliptic scalar multiplication when using NAF was given. Analogously, when using NAF on the pair of integers

Algorithm 3: Left-to-right simultaneous elliptic scalar multiplication

Input: ℓ-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

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

$$(\frac{5\ell}{9} + 2)A + \ell D \tag{7}$$

including precomputation and assuming point negation is free. This is a substantial improvement over processing the elliptic scalar multiplications separately (Equation 6 above).

Another representation was developed in [16] called *Joint Sparse Form* (JSF), which is a generalization of NAF for a pair of integers. JSF has minimal weight among all signed binary digit representations for a pair of integers, yielding an average of $\ell/2$ non-zero columns. Therefore, when using JSF Algorithm 3 requires

$$(\frac{\ell}{2} + 2)A + \ell D \tag{8}$$

which is slightly more efficient than NAF in Equation 7.

2.4 Self-Certified Keys and Signatures

Self-certified signatures 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.

A self-certified identity based (SCID) signature scheme based on the Nyberg-Rueppel signature scheme was presented in [2]. A version using groups over elliptic curves is outlined below.

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 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

$$\begin{aligned} u &\in_R \mathbb{Z}_r^* \\ (r_A, b_A) &= \text{COMPRESS}(uG) + \text{H}(ID_A) \\ s_A &= u - s_D r_A \pmod{r} \end{aligned} \tag{9}$$

and escrows[1] the private key s_A to Alice securely and values (r_A, b_A) publicly. COMPRESS is the point compression function[2], yielding the x-coordinate of uG and the compression bit b_A. 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`.

Sign. To generate the signature (c, d) on the message m, Alice uses Equation 2.

Verify. After extracting Alice's public key W_A using `Extract`, Bob verifies the signature (c, d) using Equation 3.

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

$$W_A = \text{DECOMPRESS}(r_A - \text{H}(ID_A), b_A) - r_A W_D \tag{10}$$

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.

[1] In [2], methods were presented to avoid key escrow and allow only Alice access to the secret key s_A (blinding TTP from s_A). However, this feature is beyond the scope of this paper.

[2] 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.

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

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

3 Improving the Performance of SCID Signatures

In the above SCID signature scheme, the signature is verified by using Equation 3. Therefore, simultaneous elliptic scalar multiplication can also be used in this scheme to improve performance. The extraction of the signer's public key is accomplished using Equation 10. However, these two equations can be combined to produce

$$\text{H}(m) = c - [dG + c(\text{DECOMPRESS}(r_A - \text{H}(ID_A), b) - r_A W_D)]_x \pmod{r}$$

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

$$dG + c(uG) - cr_A W_D \tag{11}$$

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.

As JSF is defined for a pair of integers, a generalization of JSF to n terms is needed. In [16], Solinas suggested this generalization as future work, with a remark questioning the practicality due to increased precomputation requirements. Such a generalization was presented independently in [13] and [7]. The need for a left-to-right method of generating the JSF was also noted, so as to work in-line with Shamir's Trick and not require separate storage of the JSF, which was presented in [14]. Algorithm 5 appearing in Appendix A is a three-term version with minor modifications[3]. This algorithm can easily be modified to perform the elliptic scalar multiplications in-line.

As seen in the previous examples, the amount of precomputation needed increases when using a signed-binary representation. Using Equation 11, 3^3 different points could be added. To compute these points,

[3] In steps 7-9b of [14], $C + 1$ columns are examined; however, only $C - 1$ columns actually need to be examined, as the left and right columns of the $N \times (C + 1)$ window are already guaranteed to have non-zero entries.

assuming negation is free, only 10 point additions actually need to be computed and stored[4]. Algorithm 4 illustrates this method.

Algorithm 4: Left-to-right three-term simultaneous elliptic scalar multiplication

Input: ℓ-bit integers j, k, l, points $P, Q, R \in E(\mathbb{F}_q)$
Output: $jP + kQ + lR$
Precompute $xP + yQ + zR \; \forall \; x, y, z \in \{-1, 0, 1\}$
Calculate 3-term JSF of j, k, l using Algorithm 5
$S \leftarrow \infty$
for $i \leftarrow \ell - 1$ **to** 0 **do**
    $S \leftarrow 2S$
    $S \leftarrow S + (j_i P + k_i Q + l_i R)$
end
return S

4 Results

A comparison of the probabilities of a non-zero column given n different scalars is presented in Table 1. These values correspond to the number of point additions needed (not including precomputation). Table 2 shows a comparison[5] of the number of required elliptic curve operations for two common standardized curves [12] when processing Equation 11 using simultaneous and separate elliptic scalar multiplications (including precomputation).

Table 1. Probabilities of a non-zero column given n terms.

n	Binary	NAF	JSF
1	.5	.3333	.3333
2	.75	.5555	.5
3	.875	.7037	.5897

[4] This assumes that point additions involving three terms (e.g. $P+Q+R$) are actually done using previous point addition results ($P+Q+R = (P+Q)+R$ using a previously computed $P + Q$ only requires one point addition).

[5] Ex Add denotes extra additions needed either for precomputation or adding resulting individual points.

Table 2. Elliptic curve operations needed for common curves.

Curve	Method	A	D	Ex Add	Field Mult	Gain
B-163	NAF (separate)	162	486	2	3256	
B-163	JSF (simul)	95	162	10	1488	**54%**
P-192	NAF (separate)	189	573	2	6418	
P-192	JSF (simul)	112	191	10	2804	**56%**

The number of field multiplications is an estimate based on using mixed coordinates. For binary curves using affine and López-Dahab coordinates, the cost of one point addition is eight field multiplications and one point doubling is four field multiplications. For prime curves using affine and Jacobian coordinates, the cost of one point addition is eight field multiplications and three field squarings and one point doubling is four field multiplications and four field squarings. Field squarings have an estimated cost of 0.85 of a field multiplication. These are generally considered the most efficient methods [8]. Equation 11 involves two fixed points (the base point and the trusted third party's public key), so these estimates could be improved by persisting the precomputed values that involve only these two fixed points.

5 Conclusion

It has been shown that using three-term JSF with three-term simultaneous elliptic scalar multiplication significantly improves performance compared to individual processing. For an elliptic curve over a binary field where $q = 2^{163}$, using three-term simultaneous elliptic scalar multiplication has over 54% less field multiplications. For an elliptic curve over a prime field where q is a 192-bit prime, there are over 56% less field multiplications. This requires minimal temporary storage and is therefore well-suited for both software and hardware implementations. These performance enhancement estimates should also be attainable in practice, as the number of field multiplications is proportional to timings of signature verifications. An application of these results was provided in the form of SC signatures, so the results also have a practical use.

6 Recent Work

When Koblitz curves are used, point doublings are replaced by an efficiently computable endomorphism called the *Frobenius map* $\tau : E(\mathbb{F}_{2^m}) \rightarrow E(\mathbb{F}_{2^m})$ such that $(x, y) \mapsto (x^2, y^2)$. τ-adic NAF [15] and two-term τ-adic

JSF [5] have been developed. For SCID signature schemes using Koblitz curves, a three-term (or more generally n-term) τ-adic JSF is needed. A generalization appears in [3].

References

1. ANSI. The elliptic curve digital signature algorithm. Technical Report ANSI X9.62-1998, American National Standards Institute, September 1998.
2. Giuseppe Ateniese and Breno de Medeiros. A provably secure nyberg-rueppel signature variant with applications. Technical Report 93, Cryptology ePrint Archive, 2004.
3. Billy Bob Brumley. Left-to-right signed-bit τ-adic representations of n integers (short paper). In *Proceedings of the 8th International Conference on Information and Communications Security (ICICS 2006)*, Raleigh, North Carolina, USA, 2006. to appear.
4. Catharina Candolin, Janne Lundberg, and Hannu Kari. Packet level authentication in military networks. In *Proceedings of the 6th Australian Information Warfare & IT Security Conference*, Geelong, Australia, November 2005.
5. 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.
6. 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.
7. 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.
8. Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer, New York, 2004.
9. Garrett Hardin. The tragedy of the commons. *Science*, 162, 1968.
10. IEEE. Standard specifications for public-key cryptography. Technical Report IEEE P1363 / D13, Institute of Electrical and Electronics Engineers, Inc., November 12 1999.
11. 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.
12. National Institute of Standards and Technology (NIST). Recommended elliptic curves for federal government use, May 1999.
13. 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.
14. 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.
15. Jerome A. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes, and Cryptography*, 19(2–3):195–249, March 2000.
16. 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.

A Three-Term JSF Algorithm

Algorithm 5: Three-Term JSF

Input: integers a', b', c', largest being of L-bit length
Output: JSF of a', b', c'
for $i \leftarrow L$ **to** 0 **do**
  $a_i \leftarrow a'_{i-1} - a'_i$ /* max 4 cols actually need to be */
  $b_i \leftarrow b'_{i-1} - b'_i$ /* examined per iteration, so the entire */
  $c_i \leftarrow c'_{i-1} - c'_i$ /* ints need not be recoded at once */
end
for $i \leftarrow L$ **to** 0 **do**
  $S \leftarrow \{a, b, c\}$ /* S holds rows with reducible bits */
  foreach $k \in S$ **do if** $k_i = 0$ **then** remove k from S
  $C \leftarrow 1$ /* C-1 0's between the non-0 bits */
  foreach $k \in S$ **do**
    for $j \leftarrow 1$ **to** 3 **do**
      if $k_{i-j} \neq 0$ **or** $j = 3$ **then**
        if $k_i + k_{i-j} \neq 0$ **then** $S \leftarrow \emptyset, C \leftarrow 1$
        else $C \leftarrow \text{MAX}(j, C)$
        break
    end
  end
  /* is there an all-0 col that should be preserved? */
  for $j \leftarrow 1$ **to** $C - 1$ **do if** $a_{i-j} = b_{i-j} = c_{i-j} = 0$ **then**
  $S \leftarrow \emptyset, C \leftarrow 1$
  /* replace $x0 \ldots 0\overline{x}$ with $0x \ldots x$ */
  foreach $k \in S$ **do**
    for $j \leftarrow 1$ **to** 3 **do**
      if $k_{i-j} = 0$ **then** $k_{i-j} \leftarrow k_i$
      else break
    end
    $k_i \leftarrow 0$
  end
  $i \leftarrow i - (C - 1)$ /* C columns have been processed */
end
return a, b, c