cogent engineering

COMPUTER SCIENCE | RESEARCH ARTICLE

Iterative sliding window method for shorter number of operations in modular exponentiation and scalar multiplication

Adamu Muhammad Noma[1], Abdullah Muhammed[1]*, Zuriati Ahmad Zukarnain[1] and Muhammad Afendee Mohamed[1,2]

Received: 27 July 2016
Accepted: 06 March 2017
First Published: 10 March 2017

*Corresponding author: Abdullah Muhammed, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, 43400, Selangor, Malaysia
E-mail: abdullah@upm.edu.my

Reviewing editor:
Duc Pham, University of Birmingham, UK

Additional information is available at the end of the article

Abstract: Cryptography via public key cryptosystems (PKC) has been widely used for providing services such as confidentiality, authentication, integrity and non-repudiation. Other than security, computational efficiency is another major issue of concern. And for PKC, it is largely controlled by either modular exponentiation or scalar multiplication operations such that found in RSA and elliptic curve cryptosystem (ECC), respectively. One approach to address this operational problem is via concept of addition chain (AC), in which the exhaustive single operation involving large integer is reduced into a sequence of operations consisting of simple multiplications or additions. Existing techniques manipulate the representation of integer into binary and *m*-ary prior performing the series of operations. This paper proposes an iterative variant of sliding window method (SWM) form of *m*-ary family, for shorter sequence of multiplications

Adamu Muhammad Noma

ABOUT THE AUTHORS

Adamu Muhammad Noma received MSc in Computer Networking at Universiti Putra Malaysia (UPM) in 2014 and is currently pursuing PhD in Computing Security at the University. His areas of research interest include networks, and computing security and optimisation. In this research, they propose a means of speeding up public key cryptosystem.

Abdullah Muhammed received PhD in computer science from the University of Nottingham in 2014. He is a senior lecturer and HoD of the Department of Communication Technology and Networks, UPM. His research interests include cloud computing, mobile and wireless network, scheduling, heuristic and optimisation.

Zuriati Ahmad Zukarnain is an associate professor at the same faculty. She received PhD from the University of Bradford. Her research interests include efficient multiparty QKD protocol for classical network and cloud; wireless and ad hoc networks load balancing; quantum processor unit for quantum computers, authentication time of IEEE 802.15.4 with multiple-key protocol, wireless networks intra-domain mobility handling, efficiency and fairness for new AIMD algorithms and kernel model.

Muhammad Afendee Mohamed received PhD in mathematical cryptography from UPM in 2011. Currently, he is a senior lecturer at this university and his research interests are in the domain of information security and mobile and wireless networking.

PUBLIC INTEREST STATEMENT

Data encryptions using public key cryptosystems provide higher functionality services such as confidentiality, authentication, integrity and non-repudiation. However, the highly computational cost in the encryption process, involving exponentiation or scalar multiplication, made this optimisation strategy relevant. In both cases, very large size of integers (of up to 600 decimal digits) are being utilised as the keys. And, the key size determines that of the exponent or the multiplier. One means of reducing the size of the computation is by transforming the process into series of simpler multiplications or additions.

In this paper, we proposed an iterative-based sliding window method (SWM) to improve the computational cost – by reducing the length of the computation. Our proposed approach searches for optimal length of the computation using the SWM, corresponding to specific integer utilized as the exponent or multiplier. The experimental results demonstrate that our proposed method improves the solution quality, by reducing the computation size, by up to 6%.

corresponding to the modular exponentiation. Thus, it is called an iterative SWM. Moreover, specific for ECC that imposes no extra resource for point negation, the paper proposes an iterative recoded SWM, operating on integers recoded using A modified non-adjacent form (NAF) for speeding up the scalar multiplication. The relative behaviour is also examined, of number of additions in scalar multiplications, with the integers hamming weight. The proposed iterative SWM methods reduce the number of operations by up to 6% than the standard SWM heuristic. They result to even shorter chains of operations than ones returned by many metaheuristic algorithms for the AC.

Subjects: Computer Science; Computing & IT Security; Algorithms & Complexity

Keywords: addition chain; modular exponentiation; scalar multiplication; sliding window method

1. Introduction

In public key cryptosystems (PKC) (Diffie & Hellman, 1976; El-Gamal, 1985; Koblitz, 1987; Rivest, Adi, & Adleman, 1978), computations involving modular exponentiation and scalar multiplication found in the respective RSA (Rivest et al., 1978) and ECC (Koblitz, 1987; Miller, 1986) are the most expensive operations that determine the efficiencies of the algorithm, and on which the security of the systems also depends. For the applications to be computationally secured the size of the key, which is the exponent or multiplier, respectively, should be of at least 1,024 bits in multiplicative structure such as Diffie–Hellman (Diffie and Hellman, 1976) and RSA and 163 bits in additive structure of ECC.

One of the means of optimizing these operations without compromising the security effectiveness is by reducing an exhaustive operation of modular exponentiation to repeated squaring and multiplication and likewise scalar multiplication to repeated doubling and addition via the concept of addition chain (AC). Since modular exponentiation is an additive function of the exponent similar to that of multiplier from scalar multiplication, both operations are adoptable to the idea of AC. In other words, possible shortening of an AC for the exponent/multiplier by reducing the number of doubling and addition corresponds to that of either one of the two operations: thus should be understood as minimizing the number of multiplications in modular exponentiation or of additions in scalar multiplication.

The problem of finding optimal AC for an arbitrary integer, also known as addition chain problem (ACP), exists for long (Dellac, 1894; Scolz, 1937). Numerous theoretical studies on the problem can be found in Balega (1976), Brauer (1939), Downey, Leong, and Sethi (1981), Mignotte and Tall (2011), Thurber (1993) and Yao (1976). From experimentation perspective, exhaustive approaches have been applied (Clift, 2010; Hatem, 2011), as well as heuristics (Bos & Coster, 1990; Gelgi & Onus, 2006; Koç, 1995; Park, Park, & Cho, 1999; Thurber, 1999). Moreover, after Downey et al. (1981) proved that the generic case called addition sequence is an NP-complete problem, various metaheuristics have also been applied (Cruz-cortés, Rodríguez-Henríquez, & Coello, 2008; Domínguez-Isidro, Mezura-Montes, & Osorio-Hernández, 2015; Jose-Garcia, Romero-Monsivais, Hernandez-Morales, Rivera-Islas, & Torres-Jimenez, 2011; León-Javier, Cruz-Cortés, Moreno-Armendáriz, & Orantes-Jiménez, 2009; Nedjah & de Macedo Mourelle, 2006; Osorio-Hernández, Mezura-Montes, Cruz-Cortés, & Rodríguez-Henríquez, 2009).

For the purpose of simplicity, a generic integer e is used in this paper to represent the exponent or multiplier of either of the operations.

Definition 1.1 Given an integer e, the sequence $a_0 = 1$, $a_1 = 2$, $a_3, \dots, a_r = e$, is said to be an AC for e if $\forall i \geq 1, a_i = a_j + a_k, i > j \geq k$. The length of the chain is r.

In the studies of AC by mean of heuristic approach, e is normally represented into an equivalent binary form, from which some form of manipulation is applied in the quest to produce the shortest possible chain.

Definition 1.2 The length of e (denoted as) $n(e)$ is defined as the minimum number of bits to represent e in binary form $e = (e_{n-1}\, e_{n-2}\, \ldots\, e_0)_2$. n used to indicate the length of an arbitrary n-bit e.

Definition 1.3 The hamming weight (shorten as weight) of e denoted as $H(e)$ is defined as the number of non-zero bits in the binary representation of e.

Using the binary form as found in many heuristic techniques, the total number of operations is counted to the number of doublings (squarings) and additions (multiplication) involved. In fact, the number of squarings is fixed to the bit-length $n(e)$, and thus improvement can only be done on the number of multiplications which is proportionate to weight $H(e)$.

Binary method (Knuth, 1998) has been the basic procedure for computing modular exponentiation as well as scalar multiplication. In the modular exponentiation, a sequence of squarings and optional multiplications are performed, depends upon the given digit value of the binary form for e that is 1 or 0, respectively. Similarly, a sequence of doublings and optional additions are performed in the scalar multiplication. For an n-bit e, represented in the binary form as $e = e_{n-1}\, e_{n-2}\, \ldots\, e_0$, the method for the exponentiation $y = x^e$ follows in Algorithm 1.

```
Algorithm 1 Binary Exponentiation
Require: $x, e = e_{n-1}e_{n-2}...e_0$
Ensure: $y \leftarrow x^e$
1: $y \leftarrow x$
2: for $j \leftarrow n-2$ to 0 do
3:     $y \leftarrow y \times y$
4:     if $e_j = 1$ then
5:         $y \leftarrow y \times x$
6:     end if
7: end for
8: return $y$
```

In Algorithm 1, $n(e) - 1$ number of squarings are performed in step 3, and a multiplication in step 5 corresponding to every non-zero bit encounter, less the most significant bit (MSB): $H(e) - 1$. Thus, assuming multiplication and squaring are computationally equal, the number of multiplications (operations) T_{bin} is

$$T_{\text{bin}}(e) = n(e) + H(e) - 2 \tag{1}$$

Since there are n bits in e, each of which is equally likely to be 1 or 0, the asymptotic number of multiplications in Algorithm 1 is $n + n/2 = 3n/2$. The method is highly efficient in implementation due to its minimal book-keeping in the process (Knuth, 1998). However, it performs excessive number of multiplications than is necessary.

An m-ary (also known as or 2^k-ary) method is an extension of the binary method. An n-bit e is padded (where necessary) with at most $k - 1$ trail of 0s to form a multiple of k. It is then partitioned into $w = \lceil n/k \rceil$ blocks of fixed k-bit words: $m_i,\ i = w-1, \ldots, 0,\ m_{w-1}$ being the most significant word (MSW). Thus, $0 \leq m_i = e_{ik+k-1}\, e_{ik+k-2}\, \ldots e_{ik} = \sum_{j=0}^{k-1} 2^j e_{ik+j} < 2^k$, and $e = \sum_{i=0}^{w-1} m_i 2^{ik}$. Initially, the values $x, x^2, \ldots, x^{2^k-1}$, corresponding to all possible value for x^{m_i}, are pre-computed. The algorithm proceed by scanning the most significant k bits m_{w-1}, raising the corresponding $x^{m_{w-1}}$ to the power of 2^k as the partial result. This is followed by subsequent scanning of the remaining $m_i,\ i = w-2, \ldots, 0$, each time multiplying the partial result by x^{m_i} and raising it to the power of 2^k as: $x^{2m_{w-1}}, x^{4m_{w-1}}, \ldots, x^{2^k m_{w-1}}, x^{2^k m_{w-1}} cdot x^{m_{w-2}}, x^{2(2^k m_{w-1} + m_{w-2})}, \ldots, x^{(2^{n-k} m_{w-1} + 2^{n-2k} m_{w-2} + \cdots + 2^k m_1)} \cdot x^{m_0} = x^e$. That is beginning from $x^{m_{w-1}}$, k-times squaring are performed followed by multiplying the partial result by the next $x^{m_i},\ i = w-2, \ldots, 0$ until the last x^{m_0} is multiplied by. The m-ary method has an average number of multiplications given by Koç (1995).

$$T_m(n,\, k) = (2^k - 2) + (\lceil n/k \rceil - 1)k + (\lceil n/k \rceil - 1)(1 - 2^k) \tag{2}$$

Depending on the *k* parameter, the method performs less number of multiplications than binary method. However, the pre-computations cost increases exponentially with an increase in the *k* size.

Note that when $m_i = 0$ ($x^{m_i} = 1$) the multiplication step is not necessary. Consequently, adaptive window methods are the enhancements of the *m*-ary that form partitions m_i of arbitrary length of 0s. In the constant-length non-zero window (CLNW) version, in the partitioning process, the leading zeros in a given *k*-bit non-zero window (NW) partition $m_i \neq 0$ are carved out and concatenated with subsequent zeros encounter to form a zero window (ZW) $m_i = 0$ partition. The ZW takes any arbitrary length until a non-zero bit is again encountered. Thus, only NWs are restricted to bit-lengths $n(m_i) \leq k$, and for which list significant bit $(LSB) = MSB = 1$. As a result, the pre-computation stage involves computing only $x^2 = x \cdot x$ and odd values $x^3 = x \cdot x^2, x^5 = x^3 \cdot x^2, \ldots, x^{2^k - 1} = x^{2^k - 3} \cdot x^2$ at the cost of 2^{k-1} multiplications. Whereas, in the variable-length non-zero window (VLNW) version (Koç, 1995), the number of ZWs is further maximized by switching NW to ZW partition construction upon encounter of predetermined $0 < q \leq k - 2$ consecutive zeros. Transition to the ZW-partition begins with the *q* zeros. Note, setting $q = k - 1$ defaults to CLNW. Both methods are also known as SWM (refer to Koç, 1995; Park et al., 1999, for details).

Given an *n*-bit *e*, partitioned into m_i windows of variable lengths such that $n(m_i \neq 0) \leq k$, the number of multiplications in $y = x^e$ is determined as follows. Beginning from the MSB, let there be $p \leq w$ number of NWs in the partition. On deferring until after the partitioning is completed and the largest NW $\max_{0 \leq i < w}(m_i)$ (henceforth denoted as $\max(m_i)$) is known, the pre-computation cost reduces to $(\max(m_i) + 1)/2$ multiplications. Beginning from MSW m_{w-1}, the exponentiation involves $n(e) - n(m_{w-1})$ squaring, and $p - 1$ multiplications corresponding to remaining NWs. The generic procedure is presented as Algorithm 2.

Algorithm 2 SWM

```
Require: $e = e_{n-1}e_{n-2}...e_0, k, q$
Ensure: $y = x^e$
Partition $e$ into NWs/ZWs, $m_i, i = w-1, \ldots, 0$, from MSB($e$) to LSB($e$):
    Carve up to $k$ incoming bits into NW until $q$ consecutive 0s are encountered;
    Carve and concatenate NW' leading 0s with all consecutive incoming 0s into ZW.
$y_1 \leftarrow x; y_2 \leftarrow x.x; y_3 \leftarrow x.y_2; y_j = y_2.y_{j-2}, j = 5, 7, \ldots, \max(m_i)$
$y \leftarrow y_{m_{w-1}}$
for $i \leftarrow w - 2$ down to 0 do
    for $j \leftarrow 1$ to $n(m_i)$ do
        $y \leftarrow y.y$
    end for
    if $m_i \neq 0$ then
        $y \leftarrow y.y_{m_i}$
    end if
end for
return $y$
```

Thus, the number of multiplications is given as follows:

$$T(e) = (\max(m_i) + 1)/2 + n(e) - n(m_{w-1}) + p - 1 \tag{3}$$

On the average, $n(m_i \neq 0)$ are maximized, while their proportionate decimal values (determined by the weight) minimized with increase in the *q* value towards *k*; the reverse is the case for relatively smaller value. On the other hand, empirical result from sets of 16 to 2,048-bit *e*s tested shows that delaying the pre-computation reduces the exponentiation cost by an average of three multiplications. Thus, given *e*, varying *k* and *q* while determining the corresponding *p*, $\max(m_i)$ and $n(\max(m_i))$ values, until those *k* and *q* that minimize Equation (3) are found, amounts to finding the optimal parameters for computing $y = x^e$ using SWM: that is it translates to finding the corresponding shortest number of multiplications in the computation. Note that only partitioning is required to determine the optimal parameters. This is the idea of the Iterative SWM.

This paper proposes finite iterative partitioning strategy to determine optimal SWM parameters for any given *e*, to achieve shortest chain of multiplications in computing modular exponentiation using the SWM. Additionally, the paper proposes an iterative version of the SWM on recoded *e* utilizing a modified NAF (Eğecioğlu & Koç, 1994): in which the increase in the NAF-length is controlled while achieving the same minimum weight. NAF is employed to reduce the cost of additions in scalar multiplication-based ECC. Furthermore the paper examined, utilizing empirical data, the relative increment in the number of additions in scalar multiplications with respect to *H*(*e*) in recoded *e*s.

The rest of this paper is organized as follows: Section 2 detailed and analysed the proposed Iterative SWM algorithm; an experiment is then set up and carried out on the algorithm, and the result discussed at the end of the section; proposed recoded version of the iterative SWM is detailed and empirically examined in Section 3. Section 4 concludes the paper.

2. Iterative sliding window method (ISWM)

The proposed ISWM utilizes left-to-right version of the VLNW algorithm (Park et al., 1999). But in this case the partition size *k* is varied from an initial k_0 to a predetermined maximum value $k_{\max}$. Similarly, the allowable consecutive zeros in a partition *q* is varied from $q_0 > 0$ to $q_{\max} \leq k - 1$. At $q = k - 1$ the algorithm is in CLNW mode (Koç, 1995). On every combination of (k, q) parameter values, the algorithm partitions *e* and determines the number of multiplications *T*(*e*) according to Equation (3). It keeps track of the parameter values with the shortest *T*(*e*), and finally evaluates and returns the corresponding SWM. The ISWM is presented as Algorithm 3.

Algorithm 3 ISWM

```
Require: $x, e, k_0, q_0, k_{\max}, q_{\max}$
Ensure: $\boldsymbol{SWM}(x, e) \leftarrow x^e$
 1: $\boldsymbol{SWMPartition}(e, k_0, q_0)$ (Call partitioning routine of the Algorithm 2)
 2: $k_{opt} \leftarrow k_0; q_{opt} \leftarrow q_0; T(e) \leftarrow (\max(m_i) + 1)/2 + n(e) - n(m_{w-1}) + p - 1$
 3: for $k \leftarrow k_0 + 1$ to $k_{\max}$ do
 4:     for $q \leftarrow q_0 + 1$ to $q_{max} < k$ do
 5:         $\boldsymbol{SWMPartition}(e, k, q)$
 6:         if $T(e) > ((\max(m_i) + 1)/2 + n(e) - n(m_{w-1}) + p - 1)$ then
 7:             $k_{opt} \leftarrow m; q_{opt} \leftarrow q;\ T(e) \leftarrow (\max(m_i) + 1)/2 + n(e) - n(m_{w-1}) + p - 1$
 8:         end if
 9:     end for
10: end for
11: return $\boldsymbol{SWM}(x, e, k_{opt}, q_{opt})$ (Call Algorithm 2)
```

2.1. Algorithm analysis

Given *n*-bit *e* and $q \leq k - 1$, in Algorithm 3, the external loop executes at most $k_{\max}$ times, in each of which the internal loop executes at most *k* times. Since $1 \geq k \geq k_{\max}$, the algorithm is bounded by $\frac{k_{\max}(k_{\max}-1)}{2} < \frac{k^2_{\max}}{2}$ number of, mainly, partitioning due to steps 1 and 3. Empirical studies (as detailed shortly) shows that optimal *k* is bounded by $O(\lg(n))$. Thus, the algorithm is bounded by $O(\lg(n)^2)$ partitionings. A complete SWM is executed once in step 11. The partitioning in step 1 is performed in a single pass, depending on the size of *k*. At worst $k = 1$ and *i*th-bit≠ $(i + 1)$th-bit, $i = n - 1, \dots, 0$: whereby the *n*-bit *e* is partitioned into $w = n$ number of m_i, thus it is bounded by *O*(*n*).

In fact, for common $n \leq 2048$ utilized in PKC, all optimal *k* value is bounded by $1 \leq k < 8$ (Koç, 1995; Park et al., 1999): Thus, at most $\frac{7\times6}{2} = 21$ partitions are needed in the process.

The memory resource required in running the algorithm is the same as that of the standard left-to-right SWM: as since partitioning is part of the SWM. Estimated as memory units required for the pre-computed and final exponent, it is at worst $2^{k-1} + O(1)$ units.

A preliminary experiment is conducted on the ISWM to estimate the bounds for the *k* and *q*, with view to optimizing the number of partitioning. Figure 1 shows the number of multiplications (*T*) as function of the $k,\ q$, corresponding to various sets of *n*.

As can be observed in Figure 1(a) and (b), and based on the empirical data collected during the experiment, the optimal *T* are (by 99.9%) within the $(k,\ q)$ parameters ranges as tabulated in Table 1.

2.2. Experimental set-up and result discussions

2.2.1. Experiment setup

Koç (1995) reported theoretical optimal parameters for the standard right-to-left SWM. The same is reviewed by Park et al. (1999). The left-to-right method is more effective in terms of shorter number of multiplications (Park et al., 1999). However, there was no exclusive report on its optimal parameters. Therefore, an empirical analysis is carried out on the method to determine the corresponding values, for classes of integers mostly utilized in PKCs. The result is presented in Table 2.

In the subsequent experiment, the number of multiplications *T* and it relative behaviour with varying weight *H*(*e*) are investigated. Thus, random sample integers are generated, to ensure adequate representation for the *e* range covered, with the details as follows:

(1) The integers are classified into 8 classes according to bit-lengths as $n = 16,\ 32,\ 64,\ 128,\ 256,\ 512,\ 1{,}024,\ 2{,}048$;

(2) Each class is divided into 16 sub-classes: 1 to 16, such that class $i{:}i = 1,\ \dots,\ 16$ has an average $H(e) = \frac{n}{32}(2i-1)$ randomly distributed in $\left[\frac{n}{16}(i-1)+1,\ \frac{n}{16}i\right]$;

(3) 1,000 sample, for each of the sub-class, is generated. And the corresponding numbers of multiplications are concurrently evaluated, applying binary method, SWM and our proposed ISWM, according to parameters in Tables 1 and 2, respectively; and

(4) The results expressed as the average of the accumulated number of multiplications for the 16,000 (1,000 × 16) sample, generated from the sub-classes, as par each class, as the representative average for the class.

2.2.2. Result discussions for ISWM

To examine the effectiveness of the ISWM in shortening the number of multiplications, an experiment was conducted using the detailed set-up and the classes of integers in Section . The results are compared with ACs (corresponding to number of multiplications) due to SWM-metaheuristic hybrid methods in Cruz-cortés et al. (2008) and Domínguez-Isidro et al. (2015). This is presented in Table 3.

In all the entries in Table 3, ISWM performs better than the standard SWM. It reduces the length of multiplications on the average by at least 2 ($n = 16$), and up to 12 ($n = 2{,}048$). This reduction is very significant considering that it is on the average basis. On the other hand, AIS-SWM and ACEP-SWM perform better than both SWM and ISWM for 128-bit integers. However, as the integer size increases, the proposed ISWM outperforms both methods: indicating superior advantage of the SWM in handling large-sized integers such as the ones used in PKCs. Based on percentage deviation of ISWM from the standard SWM, the former gains 1% shorter number of multiplications than the latter.

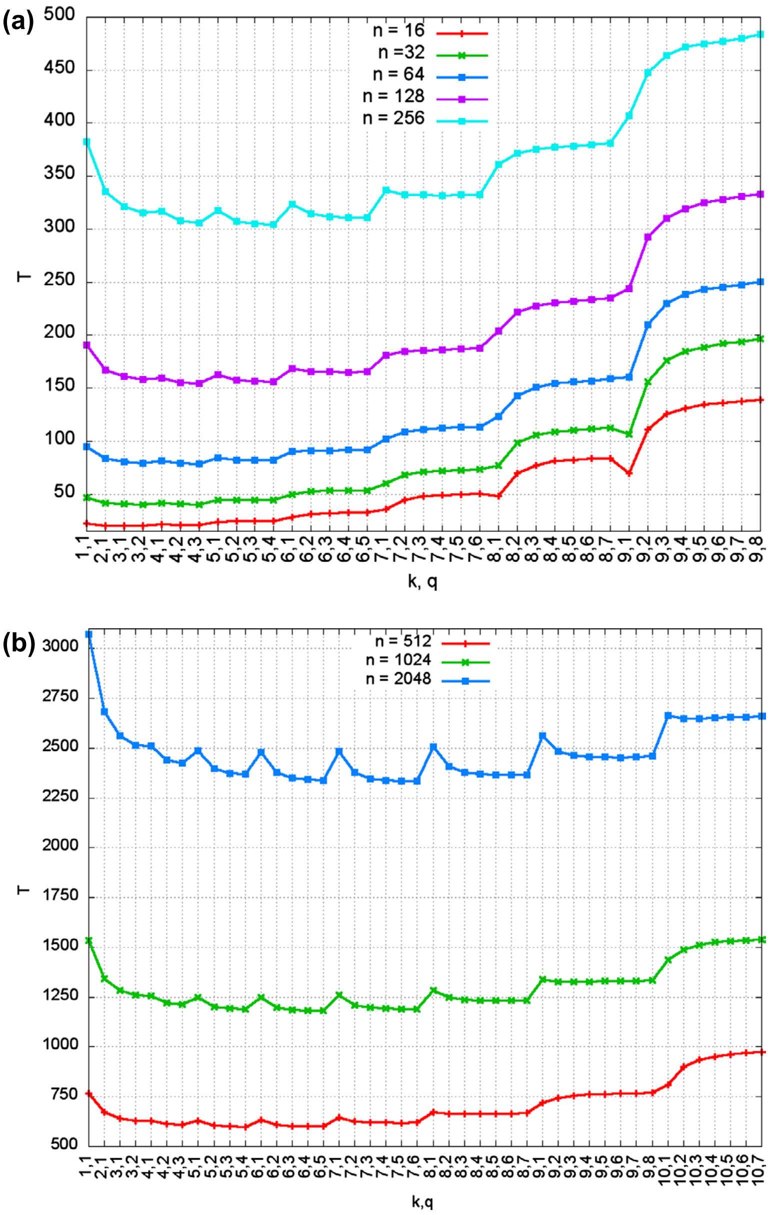

Figure 1. ISWM respond with *k*, *q* settings. (a) 16–256 bit e and (b) 512–2048 bit e.

Table 1. *k*, *q* parameters ranges for optimal ISWM

n	*k*	*q*	Partitions
16–31	2–4	1–3	6
32–255	2–5	1–4	10
256–1,023	2–6	1–4	14
1,024–2,048	2–7	1–6	21

It is interesting to note that SWM effectiveness in computing modular exponentiation with shorter number of multiplications is being underestimated at the expense of its implementation efficiency. In fact with proper choice of windows parameters, even the standard SWM is generally better than other reported heuristic/metaheuristics, for large integers (512–2,048 bits). When approached as proposed, ISWM is comparably better than the reported results in every respect. Therefore, this

Table 2. Optimal parameters for SWM

n	*k*	*q*
16, 32, 64	3	2
128, 256	4	3
512	5	4
1,024	6	5
2,048	7	6

Table 3. Average *T* by various methods

n	Bin[a]	SWM[b]	AIS-SWM[c]	ACEP-SWM[d]	ISWM
16	23	22	–	–	20
32	47	41	–	–	39
64	94	80	–	–	78
128	191	156	152	154	154
256	383	308	304	304	303
512	767	602	604	604	599
1,024	1,535	1,186	1,196	1,190	1,180
2,048	3,070	2,344	–	–	2,332

Note: "–" indicates non-available result for the given algorithm.

[a] Binary method.

[b] SWM based on Park et al. (1999).

[c] Cruz-cortés et al. (2008).

[d] Domínguez-Isidro et al. (2015).

paper concludes that SWM is still the most effective method for computing modular exponentiation, while it is second to binary method in terms of efficiency.

3. Iterative recoded SWM (IRSWM)

ECC involves repeated point additions of the form $P + P + \cdots + P$ for some finite *e* times: referred to as scalar multiplication, and denoted as *eP*. The scalar multiplication is structurally similar to modular exponentiation, with the exception that squaring and multiplication are replaced by doubling and addition, respectively. The respective operations are accordingly interchanged in this section. The ECC has the advantage of achieving equivalent level of security with shorter key size than other standard PKCs such as RSA. For example, 233-bit key ECC provides security level equivalent to 2,048-bit key RSA (Dahshan, Kamal, & Rohiem, 2015; Win, Mister, Preneel, & Wiener, 1998). Additionally, the cost of inversion is negligible: for a point $P(x,\ y)$, $P(x,\ y)^{-1} = -P(x, y) = (x,\ -y)$. Therefore, introducing inversion in the computation process is proprietary to ECC with no extra cost.

The effect of the inversion in reducing the number of additions in *eP* can be demonstrated with *n*-bit *e*s of the form $e = 2^n - 1$. They exhibit longest number of additions $2(n-1)$ on applying binary method to evaluate the corresponding $(2^n - 1)P$. But by admitting inversion, the same method reduces the additions to $n + 1$: consisting of *n* doublings and an inversion and addition, as $1P,\ 2P,\ 2^2P,\ 2^3P,\ \ldots,\ 2^nP,\ 2^nP - 1P$. In general, any *k* consecutive non-zero bits in binary form of e, $1 \times 2^{n-1} + \cdots + 0 \times 2^{n-i} + \underline{1 \times 2^{n-i-1} + 1 \times 2^{n-i-2} + \cdots + 1 \times 2^{n-i-k}} + \cdots (1 \ldots 0\underline{11 \ldots 1} \ldots)$, can be recoded into $k + 1$-bits as $1 \times 2^{n-1} + \cdots + \underline{1 \times 2^{n-i} + 0 \times 2^{n-i-1} + 0 \times 2^{n-i-2} + \cdots - 1 \times 2^{n-i-k}} + \cdots (1 \ldots \underline{10 \ldots -1} \ldots)$. Therefore, signed recoding is introduced to reduce the number of additions that follows doubling due to the integer weight $H(e)$. In this regard, balanced ternary $(-1,\ 0,\ 1)$ recoding (Knuth, 1998) is re-introduced to minimize the weight. Henceforth, -1 is denoted as $\bar{1}$ and recoded form for *e* as $\bar{e}$.

Various recoding methods exist, with NAF being canonical having the minimal non-zero bits density of *n*/3 (Eğecioğlu & Koç, 1994; Morain & Olivos, 1990; Reitwiesner, 1960). A *k*NAF is a recoded equivalent of *m*-ary, capable of reducing the density asymptotically to $n/(k+1)$ (Okeya, Schmidt-Samoa, Spahn, & Takagi, 2004). Similarly, Laih and Kuo (1997) proposed an *m*-ary version of modified signed digit (MSD), having the same non-zero bits density as the *k*NAF. Another method similar to *k*NAF, but with the advantage of performing the conversion from the MSB, is mutual opposite form (MOF) (Okeya et al., 2004). The approach eliminates the need for two parses during the conversion, and an additional *n*-bit memory required in the process is reduced to 1 (or *k*-bit for *k*MOF). Basically, for an *n*-bit *e*

$$\mathrm{MOF}(e) = (e \ll 2) \ominus e \tag{4}$$

where all the operations in Equation (4) are bitwise.

Balasubramaniam and Karthikeyan (2007) introduced complementary recoding (CR): 1 + bitwise complement of *e* is bitwise-subtracted from $2^{n(e)+1}$ such that

$$CR(e) = 2^{n(e)} \ominus \bar{e} \ominus 1. \tag{5}$$

Note that, Equation (5) is equivalent to $2^{n(e)+2} - e$. However, CR is only effective in reducing the *H*(*e*) for an *n*-bit e, when $H(e) > n/2$: This is because and its $(n+1)$-bit CR-recoded $\bar{e}$ are related as

$$H(\bar{e}) = n(e) + 1 - H(e)$$

Therefore, CR rather increases the non-zero bit density if $H(e) \leq \frac{n(e)}{2}$. For example, consider $e = 24066_{10} = 101111000000010_2$ where $H(24066) = 6$. $CR(24066) = 10\bar{1}0000\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}0_2$, $H(CR(24066)) = 10$. However, $NAF(24066) = 10\bar{1}000\bar{1}000000010$, $H(NAF(24066)) = 3$.

NAF optimally reduces *H*(*e*), but at tines leads to additional bit to *n*(*e*) that could possibly be avoided (Saffar & Said, 2015). Integers with binary form $e = 1011\ \ldots$ have NAF-recoded form as $\bar{e} = 10\bar{1}0(\bar{1}|0)\ \ldots$ having $n(\bar{e}) = n(e) + 1$. But an equal non-zero bits density can be realized without the additional 1 bit increase, by suppressing the NAF conversion at the second MSB. For example, consider $e = 93_{10} = 101110101_2$ having $n(93) = 9$ and $H(93) = 6$. $NAF(93) = 10\bar{1}00\bar{1}0101$, that is $n(NAF(93)) = 10$ and $H(NAF(93)) = 5$. On suppressing the conversion at the second MSB, 93_{10} is recoded as $1100\bar{1}0101$ with length and weight equal 9 and 5, respectively. As for suitable recoding for SWM heuristic, it is still an open problem (Win et al., 1998). Therefore, this paper proposes a modified NAF (*m*NAF) as a means of avoiding the additional bit increase where possible. The procedure is presented in Algorithm 4.

Algorithm 4 mNAF

```
Require: $e = e_{n-1}e_{n-2}, ..., e_0 : e_i = \{0, 1\}, i = n-1, ..., 0$
Ensure: $\bar{e} = e_n e_{n-1}, ..., e_0 : e_n = \{0, 1\}; e_{n-1} = 1; e_i = \{\bar{1}, 0, 1\}, i = n-2, ..., 0$
if $e_{n-2} = 1$ then
    $e_n \leftarrow 0$
end if
for $i \leftarrow 0$ to $n-1$ do
    for any $k \geq 2$ consecutive 1 bits: $e_{i+k-1}e_{i+k-2}...e_i$ do
        $e_i \leftarrow \bar{1}, e_{i+1} \leftarrow 0, ..., e_{i+k-1} \leftarrow 0, e_{i+k} \leftarrow 1$
        $i \leftarrow i + k$
    end for
end for
return $\bar{e}$
```

Table 4 presents average number of additions due to binary method, NAF and *m*NAF for sets of $10,\ 16,\ 32,\ \ldots,\ 512$-bit integers.

As can be observed in Table 4, the NAF and *m*NAF recoding significantly reduce the number of additions, especially as the integer sizes become larger. On the other hand, the proposed *m*NAF exhibits shorter additions than the original NAF: this is due to its better performance in containing length $n(\bar{e})$ while equally reducing the weight $H(\bar{e})$ as the NAF.

Iterative recoded SWM (IRSWM) is similar to ISWM (Algorithm 3). Except that *e* is initially recoded using Algorithm 4; Squaring and multiplications in SWM are replaced with doublings and additions (or subtraction), thus called recoded SWM (RSWM); and, $\max(m_i)$ is replaced with the largest absolute value $\max(|m_i|)$. The procedure is presented in Algorithm 5.

```
Algorithm 5 IRSWM
Require: $P, e = e_{n-1}e_{n-2}...e_0$
Ensure: $eP$
$\boldsymbol{mNAF}(e)$;
$\boldsymbol{SWMPartition}(e, k_0, q_0)$
$k_{opt} \leftarrow k_0; q_{opt} \leftarrow q_0; T(e) \leftarrow (\max(|m_i|) + 1)/2 + n(e) - n(m_{w-1}) + p - 1$
for $k \leftarrow k_0 + 1$ to $k_{\max}$ do
    for $q \leftarrow q_0 + 1$ to $q_{\max} < k$ do
        $\boldsymbol{SWMPartition}(e, k, q)$
        if $T(e) > (\max(|m_i|) + 1)/2 + n(e) - n(m_{w-1}) + p - 1$ then
            $k_{opt} \leftarrow k, q_{opt} \leftarrow q; T(e) \leftarrow (\max(|m_i|) + 1)/2 + n(e) - n(m_{w-1}) + p - 1$
        end if
    end for
end for
return $\boldsymbol{RSWM}(P, e, k_{opt}, q_{opt})$
```

The experimental set-up in Section 2.2.1 is utilized to test the IRSWM. However, considering that the current scalars utilized in ECC are less than 512 bits, the experiment covers 512 bits only. Likewise, little variations where observed in the optimal window parameters for the SWM, when applied on recoded integers. Accordingly, the new values are presented in Table 5.

Furthermore, preliminary experiment shows that CLNW ($q = k - 1$) on the recoded integers results in shorter number of additions than VLNW. Thus, *q* is fixed, making Algorithm 5 even faster, as the number of iterations is always less than $k_{\max}$.

Table 4. Effects of integer recoding on number of additions

n	Binary	NAF	mNAF
10	13.50 (5.5)	13.11 (4.11)	12.61 (4.11)
16	22.50 (8.5)	21.11 (6.11)	20.61 (6.11)
32	46.50 (16.50)	39.56 (8.56)	38.87 (8.56)
64	94.50 (32.50)	79.20 (16.20)	78.51 (16.20)
128	190.50 (64.50)	158.48 (31.48)	157.78 (31.48)
256	382.55 (128.55)	317.11 (62.11)	316.42 (62.11)
512	766.53 (256.53)	634.10 (123.10)	633.40 (123.10)

Note: Values in brackets show the average $H(\bar{e})$ for the respective *n*-bit *e* sets.

Table 5. Optimal parameters for RSWM

n	*k*	*q*
16	3	1
32, 64, 128	3	2
256, 512	5	4

Additionally, it is observed that level of the weight reduction due to the NAF and various other recoding methods (with the exception of CR) are subject to both the non-zero bits density and their distribution pattern: Two *n*-bit integers having the same weight exhibit different recoded density, depending on their respective pattern of the 1-bits distribution before recoding. The reduction is highly efficient when the initial 1 bits are clustered. For example, NAF for $e = 11110011101111000$ is $1000\bar{1}01000\bar{1}000\bar{1}000$, having $H(\bar{e}) = 5$. On the other hand, NAF-recoding $e = 11011011101101100$ with the same weight results to $100\bar{1}00\bar{1}000\bar{1}00\bar{1}0\bar{1}00$, having $H(\bar{e}) = 6$. As such empirical results for recoded sets of integers are highly subject to the non-zero bits distribution pattern of the set utilized. And, the resulting number of additions may not tally with theoretically expected ones, for example, asymptotic number of additions due to $NAF(n) = 4n/3$, for an *n*-bit integer. In fact, tests on integers generated by various random generators yield different results, all of which having shorter number of additions than expected theoretical one. Therefore, the results presented in this section are only subjects to the set of random integer sets utilized.

3.1. Variation in number of additions with non-zero bits density

The relative behaviour is examined, of the corresponding number of additions with respect to the non-zero bit density. The test is carried out on NAF, IRSWM and the standard SWM applied on recoded integers, RSWM. A random set of 256-bit integers is utilized. The effect of bit-length is normalized by dividing the number of additions by *n*. The result is presented in Figure 2. It also shows relative reduction in the bit-density by the methods examined.

Note that from Figure 2, initially the number of additions in both NAF and IRSWM increases proportionately with the *H*(*e*). Fortunately, peak values are reached shortly after the weight reached *n* / 2. After which the length continue to decreases. RSWM also exhibits similar trend, except that the peak value is reached when $H(e) \approx n/3$. As for the relative efficiency in containing the density, IRSWM performed best among the three methods. In general, the trend shows that integers with about 1/3–2/3 non-zero bits density exhibit relatively larger number of additions even after recoded; the reverse is the case when the density is very sparse and as well when highly concentric.

3.2. Result discussions for IRSWM

In this section, the experimental results from IRSWM are compared with that of RSWM as well as recoded binary method. The proposed *m*NAF is utilized in the recoding. The result is presented in Table 6.

Figure 2. Variation in Number of Additions with *H*(*e*) for 256-bit *e*.

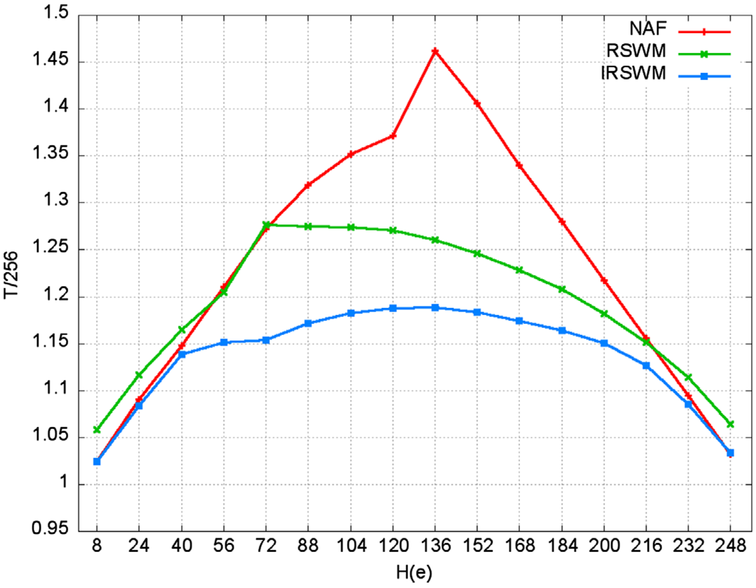

Table 6. Number of additions by various methods on recoded *e*

n	*m*NAF	RSWM	IRSWM
16	21	23	20
32	39	41	38
64	78	78	75
128	158	152	147
256	316	305	292
512	633	586	578

The result in Table 6 shows significant improvement on the number of additions returned by IRSWM over that of RSWM. It shows that iteratively searching for the optimal partitioning parameter values reduces the length, on average, by at least 2 ($n = 16$) and up to 8 ($n = 512$). As can be observed from both the Table 6 and Figure 2, IRSWM also significantly optimizes the length than the other methods examined when the integers bits densities are about half of the bit-length: which is the worst case scenario in terms of the number of additions. Overall estimate using percentage deviation shows that IRSWM improves on RSWM and SWM by 4 and 6%, respectively.

On general note: applying the IRSWM results in the shortest number of additions in ECC scalar multiplications. It is also expected that, for ECC applications in which the scalar can be chosen, the result presented may serve as a tool for much wiser and narrower selection for scalars whose number of additions is a lot lesser.

4. Conclusion and future works

This paper proposes iterative (recoded) SWM to achieve an even shorter number of operations in both modular exponentiation and scalar multiplication that are found in PKC. A modification to classic NAF algorithm is proposed, to contain the increase in integers bit-length after recoding. The relative responds is also explored, of the length of additions in computing scalar multiplications, with respect to hamming weight of the scalar. Empirical results show that the Iterative SWM optimize the number of operations by at least 1% over the SWM, and up to 6% when applied on recoded integer. With respect to relationship between the scalar multiplication and the hamming weight, recoded form of integers with original hamming weights below and above one-third of their corresponding length are more optimal for scalar multiplications than their counterparts. The paper concluded that iteratively finding the optimal window parameters while applying SWM effectively reduces the number of operation in the modular exponentiation and scalar multiplication. An even better performance is realized than the computationally much complex metaheuristic approaches at present.

Even as the proposed iterative SWM shortens the number of operations, there is still room for further optimization. It is expected that introducing simple but effective metaheuristic can further reduce operations towards (nearest) optimal.

Funding
This research was partially supported by the Malaysian Ministry of Higher Education [grant number FRGS/1/2015/ICT03/UNISZA/02/1(RR141)].

Author details
Adamu Muhammad Noma[1]
E-mail: adamnoma@yahoo.com
Abdullah Muhammed[1]
E-mail: abdullah@upm.edu.my
Zuriati Ahmad Zukarnain[1]
E-mail: zuriati@upm.edu.my
Muhammad Afendee Mohamed[1,2]
E-mail: mafendee@unisza.edu.my
[1] Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang 43400, Selangor, Malaysia.
[2] Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Besut 22200, Terengganu, Malaysia.

Citation information
Cite this article as: Iterative sliding window method for shorter number of operations in modular exponentiation and scalar multiplication, Adamu Muhammad Noma, Abdullah Muhammed, Zuriati Ahmad Zukarnain and Muhammad Afendee Mohamed, *Cogent Engineering* (2017), 4: 1304499.

References

Balasubramaniam, P., & Karthikeyan, E. (2007). Elliptic curve scalar multiplication algorithm using complementary recoding. *Applied Mathematics & Computation, 190*, 51–56.

Balega, E. G. (1976). The additive complexity of a natural number. *Soviet Mathematics Doklad, 17*, 5–9.

Bos, J., & Coster, M. (1990). Addition chain heuristics. In *Advances in Cryptology – CRYPTO' 89 Proceedings* (pp. 400–407). New York, NY: Springer.

Brauer, A. (1939). On addition chains. *Bulletin of the AMS, 45*, 736–740.

Clift, N. M. (2010). Calculating optimal addition chains. *Computing, 91*, 265–284.

Cruz-cortés, N., Rodríguez-Henríquez, F., & Coello, C. A. (2008). An artificial immune system heuristic for generating short addition chains. *IEEE Transactions on Evolutionary Computation, 12*(1), 1–24.

Dahshan, H., Kamal, A., & Rohiem, A. (2015). A threshold blind digital signature scheme using elliptic curve *dlog*-based cryptosystem. In *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)* (pp. 1–5). Glasgow: IEEE Press.

Dellac, H. (1894). Question 49. *L'Intermédiaire Math, 1*, 20.

Diffie, W., & Hellman, M. E. (1976, June). Multiuser cryptographic techniques. *Proceedings of the June 7–10, 1976, National Computer Conference and Exposition - AFIPS'76* (pp. 417–428). New York: ACM.

Domínguez-Isidro, S., Mezura-Montes, E., & Osorio-Hernández, L.-G. (2015). Evolutionary programming for the length minimization of addition chains. *Engineering Applications of Artificial Intelligence, 37*, 125–134.

Downey, P., Leong, B., & Sethi, R. (1981). Computing sequences with addition chains. *SIAM Journal on Computing, 10*, 638–646.

Eğecioğlu, O., & Koç, Ç. K. (1994). Exponentiation using canonical recoding. *Theoretical Computer Science, 129*, 407–417.

El-Gamal, T. (1985). A public key cryptosystem and a signature Scheme based on discrete logarithms. *IEEE Transaction on Information Theory, 31*, 469–472.

Gelgi, F., & Onus, M. (2006). Heuristics for minimum Brauer chain problem. In *Computer and Information Sciences – ISCIS 2006* (pp. 47–54). Heidelberg: Springer-Verlang.

Hatem, B. M. (2011). Star reduction among minimal length addition chains. *Computing, 91*, 335–352.

Jose-Garcia, A., Romero-Monsivais, H., Hernandez-Morales, C. G., Rivera-Islas, I., & Torres-Jimenez, J. (2011, October). A simulated annealing algorithm for the problem of minimal addition chains. In *Progress in Artificial Intelligence-EPIA11* (pp. 311–325). Verlang Heidelberg: Springer.

Knuth, D. (1998). Evaluation of powers. D. E. Knuth (Ed.), *The Art of Computer Programming* (Vol. 2, 3rd ed., chap. 4, pp. 461–485). Stanford, CA: Addison-Wesley.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of Computation, 48*, 203–209.

Koç, C.-K. (1995). Analysis of sliding window techniques for exponentiation. *Computers & Mathematics with Applications, 30*, 17–24.

Laih, C.-S., & Kuo, W.-C. (1997). Speeding up the computations of elliptic curve cryposystems. *Computers & Mathematics with Applications, 33*, 29–36.

León-Javier, A., Cruz-Cortés, N., Moreno-Armendáriz, M. A., & Orantes-Jiménez, S. (2009, November). Finding minimal addition chains with a particle swarm optimization algorithm. In *MICAI 2009: Advances in Artificial Intelligence* (pp. 680–691). Verlang Heidelberg: Springer.

Mignotte, M., & Tall, A. (2011). A note on addition chains. *International Journal of Algebra, 5*, 269–274.

Miller, V. (1986). Use of elliptic curves in cyptography. *Lecture Notes in Computer Science* (Vol. 218, pp. 417–428). Berlin Heidelberg: Springer.

Morain, F., & Olivos, J. (1990). Speeding up the computations on an elliptic curve using addition-subtraction chains. *Theoritical Informatics and Applications, 24*, 531–543.

Nedjah, N., & de Macedo Mourelle, L. (2006). Towards minimal addition chains using ant colony optimisation. *Journal of Mathematical Modelling and Algorithms, 5*, 525–543.

Okeya, K., Schmidt-Samoa, K., Spahn, C. & Takagi, T. (2004, August). Signed binary representations revisited. In *Advances in Cryptology – CRYPTO 2004* (pp. 123–139). Berlin Heidelberg: Springer.

Osorio-Hernández, L. G., Mezura-Montes, L., Cruz-Cortés, N., & Rodríguez-Henríquez, F. (2009, May). An improved genetic algorithm able to find minimal length addition chains for small exponents. In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation* (pp. 1–6). Trondheim: IEEE Press.

Park, H., Park, K., & Cho, Y. (1999). Analysis of the variable-length non-zero window method for exponentiation. *Computers & Mathematics with Applications, 37*, 21–29.

Reitwiesner, G. W. (1960). Binary arithmetic. *Advances in Computers, 1*, 231–308.

Rivest, R. L., Adi, S., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, 21*, 120–126.

Saffar, N. F. H. A., & Said, M. R. M. (2015). Speeding up the elliptic curve scalar multiplication using non adjacent form. *Journal of Discrete Mathematical Sciences and Cryptography, 18*, 801–821.

Scolz, A. (1937). Problem 257. *Jahresbericht der Deutschen Mathematiker-Vereinigung, 47*, 41–42.

Thurber, E. G. (1993). Addition chains - an erratic sequence. *Discrete Mathematics, 122*, 287–305.

Thurber, E. G. (1999). Efficient generation of minimal length addition chains. *SIAM Journal on Computing, 28*, 1247–1263.

Win, E. D., Mister, S., Preneel, B., & Wiener, M. (1998). On the performance of signature schemes based on elliptic curves. In *Lecture Notes in Computer Science (LNCS) 1423* (pp. 252–266). Berlin Heidelberg: Springer.

Yao, A. C.-C. (1976). On the evaluation of powers. *SIAM Journal on Computing, 5*, 100–103.