IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 2, pp. 270–285. DOI:10.46586/tches.v2023.i2.270-285

Some New Methods to Generate Short Addition Chains

Yuanchao Ding[1,2],Hua Guo[1,2,✉],Yewei Guan[1,3], Hutao Song[1,3], Xiyong Zhang[4], Jianwei Liu[1]

[1] School of Cyber Science and Technology, Beihang University, Beijing, China, {dych21,hguo,ame_reiori,htsong,liujianwei}@buaa.edu.cn
[2] State Key Laboratory of Cryptology, P.O. Box, Beijing, China,
[3] State Key Laboratory of Software Development Environment, Beihang University, Beijing, China,
[4] Beijing Institute of Satellite Information Engineering, Beijing, China, Xiyong.Zhang@hotmail.com

Abstract. Modular exponentiation and scalar multiplication are important operations in most public-key cryptosystems, and their efficient computation is essential to cryptosystems. The shortest addition chain is one of the most important mathematical concepts to realize the optimization of computation. However, finding a shortest addition chain of length r is generally regarded as an **NP**-hard problem, whose time complexity is comparable to $O(r!)$. This paper proposes some novel methods to generate short addition chains. We firstly present a Simplified Power-tree method by deeply deleting the power-tree whose time complexity is reduced to $O(r^2)$. In this paper, a Cross Window method and its variant are introduced by improving the Window method. The Cross Window method uses the cross correlation to deal with the windows and its pre-computation is optimized by a new Addition Sequence Algorithm. The theoretical analysis is conducted to show the correctness and effectiveness. Meanwhile, our experiments show that the new methods can obtain shorter addition chains compared to the existing methods. The Cross Window method with the Addition Sequence algorithm can attain 44.74% and 9.51% reduction of the addition chain length, in the best case, compared to the Binary method and the Window method respectively.

Keywords: addition chain · window method · simplified power-tree method · cross window method · addition sequence

1 Introduction

Public-key cryptosystems are widely used in practice, but they are much slower than symmetric cryptosystems. In the process of encryption and decryption, modular exponentiation and scalar multiplication are the key factors impacting the efficiency. Common public-key cryptosystems include DH, RSA, ElGamal, ECC, etc. Modular exponentiation is used in DH, RSA and ElGamal, which is

$$y = x^e mod\, c. \tag{1}$$

In ECC, scalar multiplication is used as

$$Q = eP. \tag{2}$$

In these two operations, the representation of a positive integer e as a sequence of doublings and additions is involved. In fact, the operations can be abstractly approached

Received: 2022-10-15 Accepted: 2022-12-15 Published: 2023-03-06

as an Addition Chain Problem (ACP) to find a shortest Addition Chain (AC). There are different calculation paths to get e. For example, when $e = 9$, there are 1-2-4-8-9, 1-2-3-6-9, ..., etc. Such paths are called the addition chains of e.

Given a positive integer e, an addition chain A of e with length r is a sequence of positive integers: $A = a_0, a_1, a_2, ..., a_r$, where $a_0 = 1, a_1 = 2, ..., a_r = e$, and for all $t \geq 1$, there exist i and j such that $t > i \geq j$ and $a_t = a_i + a_j$. When $i = j = t - 1$, this step is called doubling step (i.e. $a_t = 2a_{t-1}$). When $i = t - 1$, this step is called star step (i.e. $a_t = a_{t-1} + a_j, j < t$). In this paper, $n(e) = \lfloor \log_2 e \rfloor + 1$ indicates e is an n-bit integer. The Hamming weight of e, defined to be the number of 1s in the binary form of e, is denoted by $h(e)$. The shortest possible length of addition chain for e is denoted by $l(e)$.

In the practical public-key cryptosystems, the operands are usually selected with long bits for security. For example, RSA uses 1024, 2048 or 4096 bit. The operations on large integers are expensive. A shorter addition chain means faster execution of the corresponding modular exponentiation or scalar multiplication, because there is a one-to-one relationship between an element of the addition chain and the computation of modular exponentiation or scalar multiplication. For example, when $e = 9$, the addition chain 1-2-4-8-9 corresponds to $x^9 = ((x^2)^2)^2 x = x\text{-}x^2\text{-}x^4\text{-}x^8\text{-}x^9$ or $9P = 2(2(2P)) + P = P\text{-}2P\text{-}4P\text{-}8P\text{-}9P$. However, ACP is generally regarded as an **NP**-hard problem[KY00, CRJC05, NMMZ17]. Therefore, optimizating the search for a short addition chain to improve the execution speed of modular exponentiation or scalar multiplication is helpful to improve the efficiency of public-key cryptosystems.

At present, the methods of generating addition chain can be classified into two main types. The first type [BA18, BK19, TC21], called minimal length addition chain(MLAC), focuses on generating optimal addition chain, while the second one, called short addition chain(SAC), aims to generate short addition chain. The length of generated chain in the first type is minimal, which can well optimize the modular exponentiation and scalar multiplication. However, as mentioned above, the running time increases rapidly. In the second type, we can strike a balance between the length of generated chain and performance, which is more practical. In this work, we will focus on the second type of methods.

For the second type, many methods have been proposed, such as the Binary method, the m-ary (2^k-ary) method [Bra39], the Window method [Knu14], the Power-tree method [Knu14], the Genetic algorithm [CRJC05, VCR16], the Artificial Immune System [CRC08], and the Evolutionary Programming [DMO15, PCJM18], etc. The Binary method is widely used in fast modular exponentiation and scalar multiplication, which can be further optimized. The m-ary method[Bra39] divides the binary form of integers into windows, which are included in a pre-computation. Then it proceeds by scanning all the windows. The Window method[Knu14] can achieve better results than m-ary method by looking for the windows whose head and tail are non-zero, thus to reduce the pre-computation. The Power-tree method[Knu14], the Genetic Algorithm[CRJC05, VCR16], the Artificial Immune System[CRC08] and the Evolutionary Programming[DMO15, PCJM18] need complex operations, which are suitable for relatively small integers until now. The Window-based methods are feasible to solve large integers within a short time and have been repeatedly optimized.

There are a number of analyses and improvements on the Window method [BC89, KY98, KY00, AAF10, KAJK16], which are collectively called Window-based methods. In 1989, Bos and Coster[BC89] filled the pre-computation with addition sequences instead of all odd numbers, supporting bigger window size and smaller length of chain. Kunihiro and Yamamoto[KY98] further optimized the size of the pre-computation using Tunstall code. In 2000, Kunihiro and Yamamoto[KY00] proposed the Run-length method, which worked well when processing integers with large hamming weight. They also proposed the Hybrid method, a hybrid of the Run-length method and the Window method. In 2010, Mohamed et al.[AAF10] proposed an algorithm based on the Window method with small width by

using 2's Complements for finding addition-subtraction chain of 160-bit integers. In 2016, Brian et al. [KAJK16] used the Window method to find addition chain of smooth isogeny primes.

In this paper, we take into account some interesting observations about existing methods and put forward our novel ideas. To obtain a shortest addition chain, the exhaustive search of the Power-tree method is unpractical. We study the Binary method carefully and find a "backward-adding" way to construct the addition chain with our simplified power-tree, which deletes the nodes in power-tree substantially. For the Window method, adjacent windows are usually adopted which has limited window combinations. We exploit cross windows which diversify the window combinations and present our Cross Window method for better results. In the pre-computation, a new Addition Sequence Algorithm is presented to build a shorter pre-computation of the used windows, which leads to our Cross Window method with the Addition Sequence Algorithm.

To sum up, the contributions of this paper are as follows:

1. Firstly, we present a Simplified Power-tree method by deeply deleting the power-tree, whose time complexity is reduced from $O(r!)$ to $O(r^2)$.

2. Secondly, a Cross Window method is introduced by improving the Window method, which achieves better result since more window combinations are handled by using cross windows.

3. Thirdly, a Cross Window method with the Addition Sequence Algorithm is given to optimize the pre-computation in the Cross Window method with using our new Addition Sequence Algorithm, which attains 9.51% reduction of the addition chain length, in the best case, compared to the Window method.

The remaining paper is structured as follows. Section 2 briefly reviews some general existing methods. Section 3 shows a detailed description of our novel methods, including the Simplified Power-tree method, the Cross Window method and the Cross Window method with the Addition Sequence algorithm. We perform our experiments in Section 4, which shows that our methods can obtain shorter addition chains compared to the existing methods. Section 5 concludes the whole paper.

2 Existing methods

2.1 Binary Method

The Binary method (BM) uses binary representation of an integer, and an optional addition is performed depending on whether a bit is 1 or 0. The general implementation of BM is shown in Alg. 1 (from [NMMZ17]).

Algorithm 1 Binary Method

Input: $e = (e_{n-1}...e_1e_0)_2$
Output: $A = \{a_0, a_1, a_2, ..., a_r = e\}$
1: $A = \{a_0 = 1\}, r = 1$
2: **for** i from $n-2$ down to 0 **do**
3: $\quad A = A \cup \{a_r := 2a_{r-1}\}$, $r = r + 1$
4: $\quad$ **if** $e_i = 1$ **then**
5: $\quad\quad A = A \cup \{a_r := a_{r-1} + 1\}$, $r = r + 1$
6: $\quad$ **end if**
7: **end for**
8: **return** A

2.2 Power-tree Method

The Power-tree method (PTM) means that all nodes are represented in the form of a tree, and the nodes on the path are used as the addition chain of an integer. A complete power-tree without duplicate nodes on any path is a tree that contains all possible results, as shown in Fig. 1.

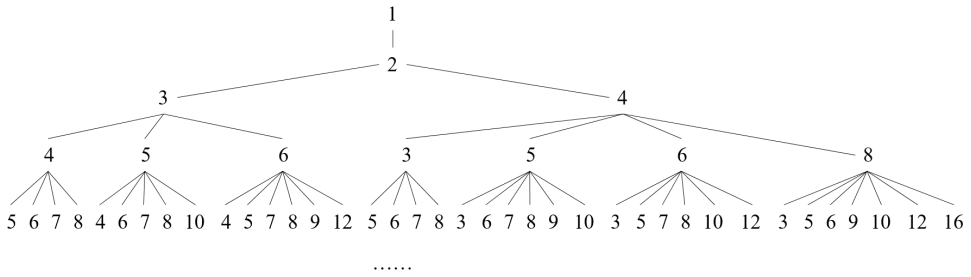

Figure 1: Power-tree.

The shortest addition chain of an integer can be determined by exhaustive search within all paths, which takes a long time. Let the depth of node 1 be 0. The number of subnodes of a node in depth r is not less than $r + 1$, and the total number of nodes in depth r is greater than $1 \cdot 2 \cdot 3 \cdot ... \cdot r = r!$.

2.3 Window Method

The idea of the Window method (WM) is to split the binary form of an integer into some windows, then the windows are processed to get the addition chain through two parts: pre-computation and construction. Let the window length be k. Pre-computation selects all odd integers from 1 to $2^k - 1$, and 2, which is $\{1, 2, 3, 5, 7, ..., 2^k - 1\}$, with length $2^{k-1} + 1$.

In WM, for the binary form of an integer e, we read a window w (the bit-length of w (denote as) $n(w) \leq k$ and $MSB(w) = LSB(w) = 1$, where $MSB(w)$ and $LSB(w)$ indicate the most and the least significant bit of w respectively). As a result, w is in the pre-computation. In construction, $n(w)$ times doubling step and one time star step are performed. For the consecutive 0s, doubling steps are directly conducted. The implementation of WM is shown in Alg. 2.

3 New methods

In this section, we propose the Simplified Power-tree Method (SPTM), the Cross Window method (CWM) and its variant the Cross Window method with the Addition Sequence algorithm (CWM-ASA). The former improves PTM by deleting the power-tree. The latters improve WM by using cross windows and optimizing the pre-computation.

3.1 Simplified Power-tree Method

We first give another view of BM. Instead of using an optional addition mixed in doubling steps, we do the optional addition when all the doubling steps are done and the addition number 1 is adjusted to corresponding numbers. This implementation of BM is Alg. 3. This view of BM shows a feasible way to construct the addition chain, which leads to the key point “backward-adding” of our Simplified Power-tree method (SPTM).

Algorithm 2 Window Method

Input: $k, e = (e_{n-1}...e_1e_0)_2$
Output: $A = \{a_0, a_1, a_2, ..., e\}$
1: $A = \{1, 2, 3, 5, 7, ..., 2^k - 1\}$
2: $i = n - 1, r = 2^{k-1}$
3: find the longest bitstring $e_ie_{i-1}...e_j$ such that $i - j + 1 \leq k$ and $e_j = 1$
4: $t = (e_ie_{i-1}...e_j)_2, i = j - 1$
5: **while** $i \geq 0$ **do**
6: **if** $e_i = 0$ **then**
7: $A = A \cup \{a_r := 2t\}, r = r + 1, t = a_{r-1}$
8: **else**
9: find the longest bitstring $e_ie_{i-1}...e_j$ such that $i - j + 1 \leq k$ and $e_j = 1$
10: **for** t from $i - j + 1$ down to 1 **do**
11: $A = A \cup \{a_r := 2t\}, r = r + 1, t = a_{r-1}$
12: **end for**
13: $A = A \cup \{a_r := t + (e_ie_{i-1}...e_j)_2\}, r = r + 1, t = a_{r-1}$
14: $i = j - 1$
15: **end if**
16: **end while**
17: **return** A

Algorithm 3 Binary Method*

Input: $e = (e_{n-1}...e_1e_0)_2$
Output: $A = \{a_0, a_1, a_2, ..., a_r = e\}$
1: $A = \{1, 2, 4, ..., 2^{n-1}\}, r = n$
2: **for** i from $n - 2$ down to 0 **do**
3: **if** $e_i = 1$ **then**
4: $A = A \cup \{a_r := a_{r-1} + a_i\}$, $r = r + 1$
5: **end if**
6: **end for**
7: **return** A

SPTM is proposed by subtly deleting tree nodes, which results in relatively small time and space complexity. A simplified power-tree consists of root chain, main chain and branch chains. The structure of the root chain is BM(m) where m is a base integer to build branch chains, and the main chain is $\{2^i | n(m) \leq i \leq t, 2^t \leq e < 2^{t+1}\}$. For each node 2^i in the main chain, a branch chain follows as $\{2^j(2^i + m) | 0 \leq j \leq t, 2^t(2^i + m) \leq e < 2^{t+1}(2^i + m)\}$. The structure of simplified power-tree is shown in Fig. 2.

Based on the simplified power-tree, the steps of constructing the addition chain of e are as follows:

(1) Obtain an addition chain of e by BM(e) and record it as the result.

(2) Search the branch chains and update the recorded addition chain whenever a shorter addition chain is found.

(3) Output the recorded addition chain.

More specifically in step (2), for the branch chain followed c_i, the corresponding addition chain of e is directly obtained if e is in the branch chain. Otherwise we form an initial chain as BM(m) $\cup$ $\{2^j | n(m) \leq j \leq i\}$ $\cup$ $\{2^j(2^i + m) | 0 \leq j \leq t, 2^t(2^i + m) \leq e < 2^{t+1}(2^i + m)\}$ and do the "backward-adding": whenever the newest integer in current chain adding the node backward in the initial chain is less than e, do the adding and append the adding result in current chain. Each branch chain can get an addition chain of e, as proved in

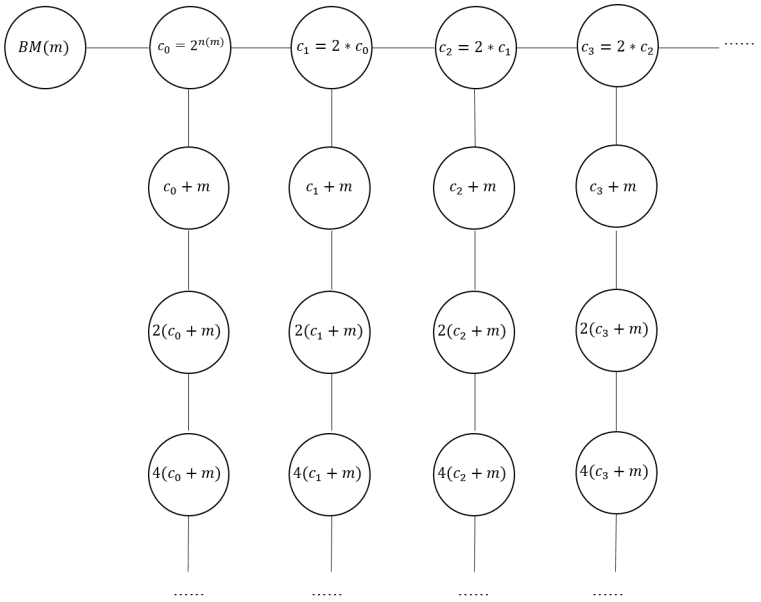

Figure 2: Simplified power-tree.

Theorem 1. Then we search all chains and update the recorded addition chain whenever we get a shorter addition chain. The implementation of SPTM is shown in Alg. 4.

Theorem 1: For any given positive integer, SPTM can produce an addition chain.

Proof: For the main chain, an addition chain of e is generated by BM. For the branch chain followed c_i, let $e = t(c_i + m) + C$, where $0 \leq C < c_i + m$. In fact, it has $0 \leq C < 2c_i - 1$ because $m < 2^{n(m)} \leq c_i$. Let $x = c_i + m$. The branch chain contains the construction of $\{x, 2x, 4x, ...\}$, which can obtain tx based on BM. That is, according to the addition chain $\mathrm{BM}(t) = \{a_0, a_1, a_2, ..., t\}$, we obtain $\mathrm{BM}_x(tx) = \{a_0x, a_1x, a_2x, ..., tx\}$. The main chain and the root chain contain $\{1, 2, 4, ..., c_i\}$, which can construct any integer from 1 to $2c_i - 1$ by BM, including C by $\mathrm{BM}(C)$. As a result, using the "backward-adding" , each branch chain can generate an addition chain of e.

Theorem 2: Let l_{SPTM} be the length of addition chain obtained by SPTM. The range of l_{SPTM} is

$$n(e) + 2\sqrt{h(e)} - 3 \leq l_{SPTM} \leq n(e) + h(e) - 2. \tag{3}$$

Proof: In the worst case, all the branch chains cannot get a shorter chain than $\mathrm{BM}(e)$. The method degenerates to BM, and the length is $n(e) + h(e) - 2$.

In the best case, all 1s in the binary form of e are divided into identical form of $2^i + m$ (denote by w), and $h(e)$ can be factorized into $h(w) \cdot \frac{h(e)}{h(w)}$. For the first window w, the length is $n(w) + h(w) - 2$. The other $\frac{h(e)}{h(w)} - 1$ windows give rise to a total of $\frac{h(e)}{h(w)} - 1$ star steps and $n(e) - n(w)$ doubling steps. Thus the addition chain length is $n(w) + h(w) - 2 + \frac{h(e)}{h(w)} - 1 + n(e) - n(w) = n(e) + h(w) + \frac{h(e)}{h(w)} - 3 \geq n(e) + 2\sqrt{h(e)} - 3$, equality holds if and only if $h(w) = \sqrt{h(e)}$.

3.2 Cross Window Method

The Window method (WM) only considers the adjacent correlation which means that each window is divided sequentially. In practice, there are windows with cross correlation which means there is a cross relationship between the windows. Using cross windows can achieve

Algorithm 4 Simplified Power-tree Method

Input: m,e
Output: $A = \{a_0, a_1, a_2, ..., e\}$

$A = BM(e)$
for i from $n(m)$ to $n(e) - 1$ **do**
  $T = BM(m) \cup \{2^{n(m)}, 2^{n(m)+1}, ..., 2^i, 2^i + m\}, r = length(T) - 1$
  while $t_r < e$ **do**
    $r = r + 1, T = T \cup \{t_r := 2t_{r-1}\}$
  end while
  if $t_r = e$ and $r < length(A)$ **then**
    $A = T$
  else
    for j from $r - 1$ down to 0 **do**
      if $t_r = e$ and $r < length(A)$ **then**
        $A = T$
      end if
      if $t_r + t_j < e$ **then**
        $r = r + 1, T = T \cup \{t_r := t_{r-1} + t_j\}$
      end if
    end for
  end if
end for
return A

a better result in somes cases. For example, for the integer $(1011111)_2$, it only needs 2 star steps using cross windows, which is less than the result using adjacent windows, as shown in Fig. 3 and Fig. 4.

1011111 101 111 1

Figure 3: Example of adjacent correlation.

10011
10011 → 1011111
1011111

Figure 4: Example of cross correlation.

The Cross Window method (CWM) is to deal with the cross windows. CWM has two parameters: valid window length k and interval expansion length s. CWM has two parts, same as WM: pre-computation and construction. In pre-computation, the valid length k is divided into two parts: the length of the right part $R = \lceil k/2 \rceil$, and the length of the left part $L = k - R \leq R$.

When $s \geq 1$, the pre-computation of CWM is obtained by inserting interval zeros between the left and right parts of the pre-computation of WM. The general binary structure is $(a)_2||0^s||(b)_2, a \in \{1, 2, 3, ..., 2^L - 1\}, b \in \{1, 3, 5, 7, ..., 2^R - 1\}$, which is performed specifically as follows:

(1) Get all odd numbers from 1 to $2^R - 1$, and 2.

(2) Get the interval expansion numbers, which are $2^R, 2^{R+1}, ..., 2^{R+s}$.

(3) Combine all numbers from 1 to $2^L - 1$ with the interval expansion and the numbers in step (1).

Finally, the pre-computation ($s \geq 1$) is $\{1, 2, 3, ..., 2^R - 1; 2^R, 2^{R+1}, ..., 2^{R+s}; 2^{R+s} + 1, 2^{R+s} + 3, ..., 2^{R+s} + 2^R - 1; 2 * 2^{R+s} + 1, 2 * 2^{R+s} + 3, ..., 2 * 2^{R+s} + 2^R - 1; ...; (2^L - 1) * 2^{R+s} + 1, (2^L - 1) * 2^{R+s} + 3, ..., (2^L - 1) * 2^{R+s} + 2^R - 1\}$, as shown by binary form in

Fig. 5.

The lengths in step (1),(2) and (3) are $2^{R-1}+1, s+1$ and $(2^L-1)2^{R-1}$. The total length is $2^{k-1}+s+2$.

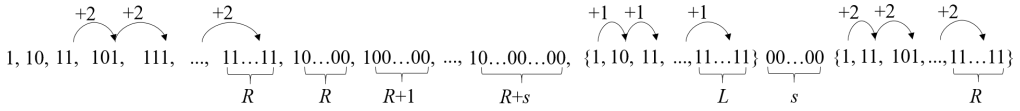

Figure 5: The pre-computation of CWM by binary form.

When the interval expansion is not carried out (i.e. $s=0$), CWM degenerates to WM. In this case, step (2) should be removed. Thus the pre-computation ($s=0$) is $\{1,2,3,5,7,...,2^k-1\}$, with length $2^{k-1}+1$, same as WM. It is unnecessary to divide the valid length k. For consistency, let $R=0$.

In the construction of CWM, for the binary form of e, we read a window w (where $n(w)\le k+s$ and $MSB(w)=LSB(w)=1$). If $n(w)>s+R$, the interval expansion positions of the window are set to 0s. If $R<n(w)<s+R$, reset w as its first R bits with removing the tail 0s. If $n(w)<R$, do nothing. As a result, w is in the pre-computation. Then an operation $\ominus$ is performed. $y\ominus x$ is to align the highest non-zero bit of y with the highest non-zero bit of x and execute a subtraction. A concrete example is listed by binary form in Fig. 6.

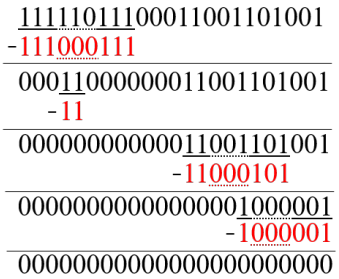

Figure 6: An example of the process of CWM ($L=R=s=3$).

For $e=(11111011100011001101001)_2$, the first window is processed as $w_0=(111000111)_2$, then do $e=e\ominus w_0=(00011000000011001101001)_2$. Repeat this for $w_1=(11)_2, w_2=(11000101)_2$ and $w_3=(1000001)_2$. Finally, e is zero. From the above example, it is easy to find that w_1 is embedded in w_0 as $(111110111)_2$. As a result, it cannot construct the addition chain like WM. To solve this problem, we record all the windows at corresponding locations and construct the addition chain from the recorded windows. That is, do doubling steps bit-by-bit from the first recorded window and add the window at each recorded position. The implementation of CWM is shown in Alg. 5.

Theorem 3: Let l_{CWM} be the length of addition chain obtained by CWM. The range of l_{CWM} is

$$n(e)+2\sqrt{h(e)}-3\le l_{CWM}\le n(e)+h(e)-2. \tag{4}$$

Proof: In the worst case, the position of 1s cannot form any window with length longer than 1, which degenerates to BM, and the length is $n(e)+h(e)-2$.

In the best case, like SPTM, all 1s are divided into several identical windows w, and the addition chain length is $n(e)+h(w)+\frac{h(e)}{h(w)}-3\ge n(e)+2\sqrt{h(e)}-3$, equality holds if and only if $h(w)=\sqrt{h(e)}$.

Algorithm 5 Cross Window Method

Input: $k, s, e = (e_{n-1}...e_1e_0)_2$
Output: $A = \{a_0, a_1, a_2, ..., e\}$

if $s \geq 1$ **then**
  $A =$ *pre-computation*$(s \geq 1)$, $R = \lceil k/2 \rceil$
else
  $A =$ *pre-computation*$(s = 0)$, $R = 0$
end if
$t = k + s, i = n - 1, M = \{\underbrace{0, 0, ..., 0}_{n}\}$
while $i \geq 0$ **do**
  if $e_i = 0$ **then**
    $i = i - 1$
  else
    find the longest bitstring $e_ie_{i-1}...e_j$ such that $i - j + 1 \leq t$ and $e_j = 1$
    if $i - j + 1 > s + R$ **then**
      $w = (e_ie_{i-1}...e_{i-R+1}\underbrace{00...00}_{s}e_{i-R-s+1}e_{i-R-s}...e_j)_2, c = i - j + 1 - s - R$
    end if
    if $R < i - j + 1 < s + R$ **then**
      find the longest bitstring $e_ie_{i-1}...e_j$ such that $i - j + 1 \leq R$ and $e_j = 1$
      $w = (e_ie_{i-1}...e_j)_2, c = i - j + 1$
    end if
    $e = e \ominus w, M_j = w, i = i - c$
  end if
end while
find the maximal j such that $M_j \neq 0$
$r = length(A), t = M_j, j = j - 1$
while $j \geq 0$ **do**
  $A = A \cup \{a_r := 2t\}, r = r + 1, t = a_{r-1}$
  if $M_j \neq 0$ **then**
    $A = A \cup \{a_r := t + M_j\}, r = r + 1, t = a_{r-1}$
  end if
  $j = j - 1$
end while
return A

Theorem 4: In general, let the number of recorded windows be v, the length of pre-computation be $2^{k-1} + s + 1 + \beta$ ($\beta = 0$ if $s = 0$ otherwise $\beta = 1$), and let the first window be w_0, the addition chain length obtained by CWM is

$$l_{CWM} = 2^{k-1} + s + \beta + n(e) - n(w_0) + v. \quad (5)$$

Proof: In CWM, we first construct the pre-computation. The length of pre-computation is $2^{k-1}+s+2$ if $s \geq 1$ otherwise is $2^{k-1}+1$, i.e. $2^{k-1}+s+1+\beta$ where $\beta = 0$ if $s = 0$ otherwise $\beta = 1$. Then perform $n - n(w_0)$ times doubling step repeatedly and $v - 1$ times star step for recorded windows except the first window. Thus the addition chain length obtained by CWM is $l_{CWM} = 2^{k-1} + s + 1 + \beta + n(e) - n(w_0) + v - 1 = 2^{k-1} + s + \beta + n(e) - n(w_0) + v$.

3.3 Cross Window Method with Addition Sequence Algorithm

The pre-computation of CWM can be optimized since some integers in the pre-computation may not be used as a window. In this paper, a new Addition Sequence Algorithm (ASA)

is presented to construct a short pre-computation of the used windows. Addition Sequence (AS) refers to the shortest addition chain containing given multiple integers, which is an **NP**-complete problem. However, AS is solvable in CWM, since only the pre-computation is involved which contains small integers. When we obtain a shorter pre-computation, we can also use larger valid window length and interval expansion length and are possible to obtain shorter addition chain.

Now we give a pragmatic ASA, which can find a short addition chain containing all the used windows quickly. For an increasing order sequence $A = \{e_0, e_1, ..., e_{d-1}\}$, let the last two numbers be $x, y(y > x)$ and let $y = tx + C(0 \leq C < x)$. For tx, we get $\mathrm{BM}_x(tx) = \{a_0x, a_1x, a_2x, ..., tx\}$ and put it in A by increasing order. We put C in A by increasing order if it is not in A and is non-zero. Thus the addition chain from x to y is formed. Repeat the above steps for the following two numbers in A in reverse order until all integers in A are solved. Finally, an addition chain containing $e_0, e_1, ..., e_{d-1}$ is obtained. The implementation of ASA is shown in Alg. 6.

Algorithm 6 Addition Sequence Algorithm

Input: $A = \{e_0, e_1, ..., e_{d-1}\}$
Output: *an addition chain containing all integers in A*

```
1: arrange A in increasing order
2: add 1, 2 into A in increasing order if they are not in A
3: j = length(A) − 1
4: while j > 1 do
5:     y = a_j, x = a_{j−1}, y = tx + C(0 ≤ C < x)
6:     add BM_x(tx) into A in increasing order
7:     if C ≠ 0 and C ∉ A then
8:         add C into A in increasing order
9:         j = j + 1
10:    end if
11:    j = j − 1
12: end while
13: return A
```

When the result of ASA is shorter than the original pre-computation in CWM (satisfied in most cases, but not absolutely), the original pre-computation will be replaced. CWM with ASA (CWM-ASA) is implemented in Alg. 7.

Algorithm 7 Cross Window method with the Addition Sequence Algorithm

Input: *k,s,e*
Output: $A = \{a_0, a_1, a_2, ..., e\}$

```
1: A = CWM(k, s, e), let all the windows used be a sequence N
2: ASA-chain = ASA(N)
3: if length(ASA-chain) < length(pre-computation) then
4:     replace the pre-computation in A with ASA-chain
5: end if
6: return A
```

Theorem 5: Let $l_{CWM\text{-}ASA}$ be the length of addition chain obtained by CWM-ASA, the range of $l_{CWM\text{-}ASA}$ is the same as l_{CWM}, which is

$$n(e) + 2\sqrt{h(e)} - 3 \leq l_{CWM\text{-}ASA} \leq n(e) + h(e) - 2. \tag{6}$$

Proof: The difference between CWM-ASA and CWM is the pre-computation. The

lower bound of the pre-computation length in CWM-ASA is exactly the pre-computation length in CWM, thus the range of $l_{CWM\text{-}ASA}$ is the same as l_{CWM}.

Theorem 6: In general, let the number of recorded windows be v, the length of pre-computation be u, and the first window be w_0, the addition chain length obtained by CWM-ASA is

$$l_{CWM\text{-}ASA} = u + n(e) - n(w_0) + v - 1. \tag{7}$$

Proof: In CWM-ASA, we first construct the pre-computation with length u and then perform $n - n(w_0)$ times doubling step repeatedly and $(v-1)$ times star step for recorded windows except the first window. Thus the addition chain length obtained by CWM-ASA is $l_{CWM\text{-}ASA} = u + n(e) - n(w_0) + v - 1$.

4 Numerical Results

In this section, we implement BM, WM, SPTM, CWM and CWM-ASA and the performance are compared. We firstly show the performance of these methods on small integers with $l \leq 22$. Then a general case is conducted with the integers generated randomly with different Hamming weight. Moreover, the integers of effective types of SPTM are exhibited to indicate the irreplaceable advantages of SPTM in some cases. The parameters are selected as: WM: $1 \leq k \leq 20$; SPTM: $1 \leq m \leq 63$ and m is odd; CWM: $1 \leq k \leq 10, 0 \leq s \leq 10$; CWM-ASA: $1 \leq k \leq 20, 0 \leq s \leq 20$. The final result for an integer of a method is the shortest addition chain length within the parameter range.

4.1 The Integers with $l \leq 22$

For 365634 positive integers with $l \leq 22$ [Cli], the results of BM, WM, SPTM, CWM and CWM-ASA are shown in Table 1.

Table 1: Results of all integers with $l \leq 22$.

Gap with the	BM		WM		SPTM		CWM		CWM-ASA	
shortest	count	prop.	count	prop.	count	prop.	count	prop.	count	prop.
0	7880	0.0216	46193	0.1263	68586	0.1876	86565	0.2368	228805	**0.6258**
1	31480	0.0861	150463	0.4115	187621	0.5131	185261	0.5067	131440	0.3595
2	66045	0.1806	127654	0.3491	100286	0.2743	83943	0.2296	5379	0.0147
3	81569	0.2231	37075	0.1014	9051	0.0248	9597	0.0262	10	2.7E-5
4	74605	0.2040	4231	0.0116	90	0.0002	267	0.0007	0	0.0000
5	59299	0.1622	18	4.9E-5	0	0.0000	1	3.0E-6	0	0.0000
6	26668	0.0729	0	0.0000	0	0.0000	0	0.0000	0	0.0000
7	13672	0.0374	0	0.0000	0	0.0000	0	0.0000	0	0.0000
8	3334	0.0091	0	0.0000	0	0.0000	0	0.0000	0	0.0000
9	893	0.0024	0	0.0000	0	0.0000	0	0.0000	0	0.0000
10	136	0.0004	0	0.0000	0	0.0000	0	0.0000	0	0.0000
11	52	0.0001	0	0.0000	0	0.0000	0	0.0000	0	0.0000
12	1	3.0E-6	0	0.0000	0	0.0000	0	0.0000	0	0.0000
AVG	3.5433		1.4605		1.1369		1.0475		**0.3890**	

In this range, from the first row, we can see the optimal results proportions of BM, WM, SPTM, CWM and CWM-ASA are 2.16%, 12.63%, 18.76%, 23.68% and 62.58%

respectively, and from the last row the average gap with the shortest is 3.5433, 1.4605, 1.1369, 1.0475, and 0.3890 respectively. The results of SPTM, CWM and CWM-ASA are better than those of BM and WM, and are more concentrated on the part with smaller gap. CWM-ASA has the best results, and the optimal and suboptimal (the gap with the shortest is 1) results account for 98.53%.

4.2 The Integers Generated Randomly with Different Hamming Weight

Let $p = \frac{Hamming\ weight}{bit\text{-}length}$, which means the bit 1 occurs with the probability of p. Select bit-length as 160, 384, 512, 1024, 2048, 4096 and p as 0.1, 0.2, 0.4, 0.5, 0.6, 0.8, 0.9. Set 50 integers for each combination. The average addition chain lengths are shown in Table 2 by bit-length.

Table 2: Average addition chain lengths of random integers by bit-length.

Len/bit	BM	WM	SPTM	CWM	**CWM-ASA**
160	238.15	192.37	210.15	191.93	**187.89**
384	574.87	452.64	513.37	452.14	**445.27**
512	766.47	600.18	688.10	599.64	**590.71**
1024	1534.46	1182.50	1388.86	1181.94	**1166.94**
2048	3070.68	2335.35	2793.60	2334.88	**2307.24**
4096	6139.43	4620.42	5610.92	4619.92	**4567.07**

In the random case, for each bit-length, the results obtained by SPTM is greater than WM, which shows that SPTM is not effective in this case. The results obtained by CWM and CWM-ASA are better compared to WM, and the chain length obtained by CWM-ASA is relatively short. Fig. 7 shows the chain length optimization degree of CWM-ASA compared with WM.

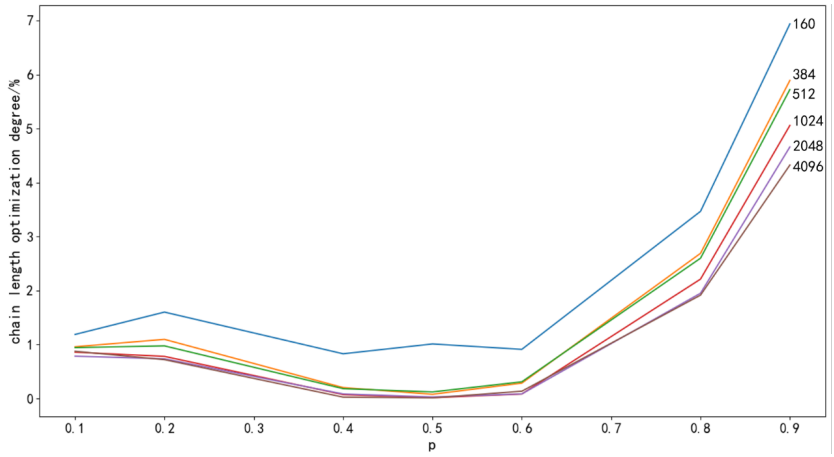

Figure 7: Chain length optimization degree of CWM-ASA compared with WM.

When $p \leq 0.5$, the optimization degree generally declines with the increasing of p, and the overall optimization degree is relatively low; when $p = 0.5$, the optimization degree is approximately the lowest; when $p \geq 0.5$, the optimization degree generally increases with the increasing of p, and the overall optimization degree is relatively high.

For the bit-length, with the increasing of the bit-length, the optimization degree of CWM-ASA declines. This is because the corresponding extra times doubling step are

unavoidably brought in with the increasing of the bit-length, so that the overall cardinality becomes larger.

In addition, the numbers with larger Hamming weight ($p = 0.95$) are tested, as shown in Table 3. For $p = 0.95$, the average optimization degree of CWM-ASA is 43.11% compared with BM. When the bit-length is 4096, the average addition chain length obtained by BM is 7988.70, while CWM-ASA is 4414.82, and the optimization degree reaches 44.74%. The average optimization degree of CWM-ASA is 7.89% compared with WM. When the length is 160 bit, the average addition chain length obtained by WM is 202.06, while CWM-ASA is 182.84, and the optimization degree reaches 9.51%.

Table 3: Comparison of CWM-ASA and BM, WM when $p = 0.95$.

Len/bit	BM	WM	CWM-ASA	optimization (CWM-ASA & BM)	optimization (CWM-ASA & WM)
160	310.32	202.06	182.84	41.08%	9.51%
384	746.76	470.42	430.36	42.37%	8.52%
512	997.14	622.04	569.98	42.84%	8.37%
1024	1994.92	1218.78	1126.40	43.54%	7.58%
2048	3989.96	2394.72	2230.84	44.09%	6.84%
4096	7988.70	4724.02	4414.82	44.74%	6.55%

4.3 The Integers of Effective Types for SPTM

SPTM is effective to the integers which have windows whose higest bit are followed by a long series of 0s and the rest of the windows. In this case, the length of the window is so long that the pre-computations of WM and CWM are overwhelming. Without using pre-computation, the result of SPTM is better.

The generation rules of test integers are as follows:

(1) Randomly select $4 \leq k \leq 6$ and $40 \leq s \leq 60$.

(2) The integers of k bits is generated randomly, and then one bit 1 and s 0s are set ahead to form a window, and the window is copied to $k + 1$ copies.

(3) The positions of these windows are randomly generated, and the position distance among the windows is not less than k. Then an integer is obtained from these windows with removing the tail 0s.

The average test results are shown in Table 4.

Table 4: Average test results of effective types for SPTM.

Len/bit	BM	WM	SPTM	CWM	CWM-ASA
160	167.10	166.24	**163.28**	165.48	165.34
384	395.52	393.34	**388.74**	392.60	392.42
512	523.38	521.30	**516.70**	520.60	520.44
1024	1035.12	1033.08	**1028.64**	1032.56	1032.44
2048	2059.10	2057.38	**2052.62**	2056.62	2056.54
4096	4107.90	4105.26	**4100.78**	4104.78	4104.76

In this case, the results obtained by SPTM are the best, and the results obtained by CWM and CWM-ASA are also better than those obtained by WM. This shows that

although SPTM is not suitable for the random integers, it can achieve the best results among several methods for the windows having the higest bit followed by a considerable number of 0s.

4.4 Computational Cost and Memory Usage

In SPTM, for any given positive integer e, because the main chain and branch chains mainly contain doubling steps, their lengths are approximately equal to the bit-length of e (i.e. $O(\log e)$). The number of branch chains is also $O(\log e)$ and a total of $O((\log e)^2)$ additions are performed. Thus, the time complexity of SPTM is $O((\log e)^2)$. The branch chains are searched one-by-one and the recorded addition chain is constantly updated, so that the space complexity is $O(\log e)$.

In CWM and CWM-ASA, the computational cost and memory usage mainly come from the generation of the obtained addition chain. The computational cost and memory usage of the pre-computation, the window locations and the chain obtained by ASA are negligible, since they handle small integers as windows. The length of the obtained addition chain is $O(\log e)$ and each element is generally added by a doubling step or a star step. Thus, for CWM and CWM-ASA, the time complexity and the space complexity are all $O(\log e)$.

We give an estimate of the memory usage of the proposed methods in Table 5. For an addition chain of e, each element in the addition chain needs at most $n(e)$ bits of memory usage and the total memory usage is approximately $(n(e))^2$ bits. SPTM needs twice memory usage because SPTM stores the current chain and the recorded chain. The proposed methods can be performed in a short time. SPTM can complete the computation in 1 second when the bit-length of the target integer is less than 2048 and in several seconds for the integers of 4096 bits. CWM and CWM-ASA only need several milliseconds for computation, even for the integers of 4096 bits.

Table 5: Memory usage (in KiB) of the proposed methods.

Len/bit	SPTM	CWM	CWM-ASA
160	6.25	3.13	3.13
384	36	18	18
512	64	32	32
1024	256	128	128
2048	1024	512	512
4096	4096	2048	2048

5 Conclusion

In this paper, we proposed a Simplified Power-tree method and a Cross Window method with a new Addition Sequence algorithm. The Simplified Power-tree method constructs a power-tree with deep deletion, which is more suitable when the windows have the highest bit followed by a considerable number of 0s. The Cross Window method considers the windows with cross relationship. The cross windows are processed by recording the window positions for recovery. Furthermore, the pre-computation is optimized with the Addition Sequence Algorithm. The Cross Window method is slightly better than the Window method, and the Cross Window method with the Addition Sequence algorithm has a better optimization, especially in the case of large Hamming weight. Roughly speaking, the average optimization degree is 7-8%, and the best case is 9-10%.

Acknowledgment

We would like to thank the anonymous reviewers for their helpful and constructive comments. This paper is supported by the National Key R&D Program of China (2021YFB2700200), Natural Science Foundation of Beijing Municipality (No. 4202037), the National Natural Science Foundation of China (U21B2021, 61972018, 61932014).

References

[AAF10] Mohamed M. Abd-Eldayem, Ehab T. Alnfrawy, and Aly A. Fahmy. Addition-subtraction chain for 160 bit integers by using 2's complement. *Egypt. Comput. Sci. J.*, 34(5), 2010.

[BA18] Hatem M. Bahig and Khaled A. AbdElbari. A fast gpu-based hybrid algorithm for addition chains. *Clust. Comput.*, 21(4):2001–2011, 2018.

[BC89] Jurjen N. Bos and Matthijs J. Coster. Addition chain heuristics. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 400–407. Springer, 1989.

[BK19] Hazem M. Bahig and Yasser Kotb. An efficient multicore algorithm for minimal length addition chains. *Comput.*, 8(1):23, 2019.

[Bra39] Alfred Brauer. On addition chains. *Bulletin of the American mathematical Society*, 45(10):736–739, 1939.

[Cli] N. Clift. Shortest addition chains. http://wwwhomes.uni-bielefeld.de/achim/addition_chain.html.

[CRC08] Nareli Cruz Cortés, Francisco Rodríguez-Henríquez, and Carlos A. Coello Coello. An artificial immune system heuristic for generating short addition chains. *IEEE Trans. Evol. Comput.*, 12(1):1–24, 2008.

[CRJC05] Nareli Cruz Cortés, Francisco Rodríguez-Henríquez, Raúl Juárez-Morales, and Carlos A. Coello Coello. Finding optimal addition chains using a genetic algorithm approach. In *Computational Intelligence and Security, International Conference, CIS 2005, Xi'an, China, December 15-19, 2005, Proceedings, Part I*, volume 3801 of *Lecture Notes in Computer Science*, pages 208–215. Springer, 2005.

[DMO15] Saúl Domínguez-Isidro, Efrén Mezura-Montes, and Luis Guillermo Osorio-Hernández. Evolutionary programming for the length minimization of addition chains. *Eng. Appl. Artif. Intell.*, 37:125–134, 2015.

[KAJK16] Brian Koziel, Reza Azarderakhsh, David Jao, and Mehran Mozaffari Kermani. On fast calculation of addition chains for isogeny-based cryptography. In Kefei Chen, Dongdai Lin, and Moti Yung, editors, *Information Security and Cryptology - 12th International Conference, Inscrypt 2016, Beijing, China, November 4-6, 2016, Revised Selected Papers*, volume 10143 of *Lecture Notes in Computer Science*, pages 323–342. Springer, 2016.

[Knu14] Donald E Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[KY98] Noboru Kunihiro and Hirosuke Yamamoto. Window and extended window methods for addition chain and addition-subtraction chain. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 81(1):72–81, 1998.

[KY00] Noboru Kunihiro and Hirosuke Yamamoto. New methods for generating short addition chains. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 83(1):60–67, 2000.

[NMMZ17] Adamu M. Noma, Abdullah Muhammed, Mohamad Afendee Mohamed, and Z. Ahmad Zulkarnain. A review on heuristics for addition chain problem: Towards efficient public key cryptosystems. *J. Comput. Sci.*, 13(8):275–289, 2017.

[PCJM18] Stjepan Picek, Carlos A. Coello Coello, Domagoj Jakobovic, and Nele Mentens. Finding short and implementation-friendly addition chains with evolutionary algorithms. *J. Heuristics*, 24(3):457–481, 2018.

[TC21] Edward G. Thurber and Neill Michael Clift. Addition chains, vector chains, and efficient computation. *Discret. Math.*, 344(2):112200, 2021.

[VCR16] Eduardo Vázquez-Fernández, Carlos Cadena, and David A. Reyes-Gomez. A genetic algorithm with a mutation mechanism based on a gaussian and uniform distribution to minimize addition chains for small exponents. In *IEEE Congress on Evolutionary Computation, CEC 2016, Vancouver, BC, Canada, July 24-29, 2016*, pages 935–940. IEEE, 2016.