第 37 卷第 1 期 重庆工商大学学报(自然科学版) 2020 年 2 月
Vol. 37 No. 1 J Chongqing Technol & Business Univ(Nat Sci Ed) Feb. 2020

doi:10.16055/j.issn.1672-058X.2020.0001.002

NAF 的二进制表示法及其算法研究*

蒋洪波[1],孙 宇[2],张鹏南[2],冯新宇[1],王明杰[3]

(1. 黑龙江科技大学 电子与信息工程学院,哈尔滨 150022; 2. 工业和信息化部电子第五研究所,广州 510610 ;
3. 哈尔滨煤矿机械研究所,哈尔滨 150036)

摘 要:椭圆曲线加密的快速实现研究一直是该领域的研究热点,其中二进制数的非相邻表示型(NAF)因此被广泛应用,它主要应用在点乘运算,在该算法中用到的 NAF 是由带符号位的数字组成,所以通常采用一位一存储的方式,然而在一些存储资源有限的设备上这是极大的浪费;为了节省存储资源,提出一种 NAF 的二进制表示方法,这样就能将多位 NAF 数值按照运行平台的字长来存储,大大提高了存储资源的利用率;在此基础上给出 NAF 二进制表示法的算法及其点乘算法;实验结果表明该表示法的运算效率较原算法的效率没有太大的影响,尤其在点乘运算中影响更是微弱,但是在提高存储效率方面表现突出,节省存储空间达 96% 以上。

关键词:非相邻表示型;二进制表示法;点乘

中图分类号:TN918.4 **文献标志码**:A **文章编号**:1672-058X(2020)01-0008-06

0 引 言

椭圆曲线有着优良的群结构,因此被广泛应用。自从 Diffie 和 Hellman 提出了公钥密码体系的思想后,Kolblitz 和 Miller 分别将其应用到公钥密码,其在公钥密码应用上也得到了快速发展, 当时的研究成果主要集中在安全性和有效性方面。近年来随着生活节奏的加快和计算机的极速发展,人们将其使用到移动设备上的例子越来越多,因此关注热点就转移到快速实现和节约资源上来了。由于时间复杂度和空间复杂度是不可调和的两方面,所以在快速实现上人们往往使用牺牲存储资源的方式,例如:点乘运算用预计算的方式存储个别点乘结果,用查表的方式快速取得相应数值进行计算。然而为了节约成本,很多移动设备上的资源是有限的,这就限定了预计算的实现方式。在文献[1]中给出了用二进制 NAF(Non-adjacent Form,非相邻表示型)方法计算点乘的算法,该算法不需要预计算,但前提是需要将进行点乘的 k 做 NAF 表示。通常在做 NAF 计算时其结果是按照 k 的二进制表示一位一位计算并存放的,这样做计算简单,利用起来也方便,然而这种方法浪费了大量的存储资源。试想若 k 为 163 位的二进制数,那么就需要 163 个存储单元将其存储起来,如果实现设备的存储字长为 W 位,那么就浪费了至少 $163\times(W-2)$ 位的资源。

基于 NAF 的性质研究其组成特点,提出了一种用二进制表示的 NAF 方法,这种方法可以将常用 NAF 的 3 种数值(-1,0,1)表示成二进制数,用字长 W 的字来存储,若 k 的 NAF 为 163 位,则使用 $163/W$ 个单元即可存放,以 $W=32$ 为例,能节约 96% 左右的存储单元。

1 非相邻表示型(NAF)

一个正整数 k 可以使用带符号的二进制表示,

收稿日期:2019-06-27;修回日期:2019-08-11.

* 基金项目:黑龙江省教育厅面上指导项目资助(12541715).

作者简介:蒋洪波(1978—),女,黑龙江省鸡西人,副教授,硕士,从事 SoC 和椭圆曲线加密研究.

即 $k=\sum_{i=0}^{l-1}k_i 2^i$,其中 $k_i \in \{-1,0,1\}$, $k_{l-1}\neq 0$,1 为 NAF 的长度,且不存在非零的两个连续数字 k_i。

文献[1]的定理 3.29 给出了 NAF 的 5 条性质。

定理 1　设 k 是一个正整数:

(1) k 有唯一的非相邻表示型,记作 $NAF(k)$;

(2) k 的 $NAF(k)$ 具有最少的非零数;

(3) $L \leqslant l \leqslant L+1$, L 为 k 的二进制表示长度;

(4) $\frac{2^l}{3}<k<\frac{2^{l+1}}{3}$;

(5) $NAF(k)$ 的非零数字平均密度约为 1/3。

计算 $NAF(k)$ 的每位数字,通常用 k 连续除以 2 得到,除得结果取余数。如果 k 为奇数,那么余数 r 取值-1 或 1,使得 $(k-r)/2$ 是偶数,则下一个 NAF 的值为 0。文献[1]的算法 3.30 给出了计算一个正整数 k 的 NAF 算法。文献[2]对该算法进行了改进,使得算法的运算在速度上有所提高。

基于以上算法可以得到 $k=55$ 和 $k=48$ 的 NAF 表示,具体表示细节如表 1 所示。

表 1　整数 k 的 NAF 对照表

Table 1　Comparison table of NAF for integer k

k 值	NAF 值
48	1 0 -1 0 0 0 0
55	1 0 0 -1 0 0 -1

从表 1 可以看出 NAF 的数值只有 3 种形式:1、0 和-1,若想存储该 NAF 值最直接简单的方式就是用一个长度为 l 的数组存放,也恰好符合每次循环求得 1 位 NAF 值的规律,基于该算法的点乘算法也不需要预计算,直接从左到右判断每位 NAF 的值,是 1 就加 P,是-1 就减 P,每次加减 P 前都做倍点运算,具体算法详见文献[1]的算法 3.31。

2　NAF 的二进制表示法及其相关算法

若采用每位 NAF 值占用一个数组单元的方式存储数值,会造成资源的极大浪费,但是注意到 NAF 的取值有“-1”这一数值,由于它是带符号的无法像普通二进制那样存储,所以想用一个字节来存放多个 NAF 数值变得不太可能。

根据文献[1]中定理 3.29 的性质,又观察表 1 的每个 NAF 值,由于不存在非零的两个连续数字 k_i,也就是说两个非零数字之间至少有一个零存在,因此试想可以将“-1”用符号位的形式表示成“11”,其他位保持不变,即 0 和 1 保持不变,那么-1 表示成的 11 只要多占用一位 0 的位置即可,也就是说 11 占用-1 的左边或右边的 0 位置,原来的 0 去掉。这样一来,整个 NAF 序列里就都是 0 和 1 的数值了,也就为用字存储 NAF 提供了可能。

涉及具体做法要考虑-1 的 11 表示是占用左边的 0 还是右边的 0,这就要根据具体用法来定了。由于在做点乘时采用的是从左到右的二进制点乘方法,因此这里给出-1 向右扩展的方法。具体做法可以通过表 2 进行对比。

表 2　k 的 NAF 值和 NAF 的二进制表示法对照表

Table 2　Comparison table of binary representation value for NAF and NAF for integer k

k 值	二进制值	二进制位宽	NAF 值	NAF 值位宽	NAF 的二进制表示法	NAF 的二进制表示法位宽
48	1 1 0 0 0 0	6	1 0 -1 0 0 0 0	7	1 0 1 1 0 0 0	7
55	1 1 0 1 1 1	6	1 0 0 -1 0 0 -1	7	1 0 0 1 1 0 1 1	8

表 2 中“1 0 0 -1 0 0 -1”由于最后 1 位是“-1”,因此遵循向右扩展原则要向后扩展 1 位,所以造成表中位宽由原来的 7 位变成了 8 位。

2.1　NAF 的二进制表示法算法

假设字宽为 W, l 为最后计算 NAF 的位长,则 NAF 的向右扩展二进制表示法算法如下所示。

算法 1　NAF 的向右扩展二进制表示法

输入:正整数 k。

输出:$(A[j], A[j-1], \cdots, A[0])$。

Step 1　数组 A 初始值为 0,变量 t、i、j 分别初始化为 0;

Step 2　$k \geqslant 1$ 时,重复执行

(1) 若 $i=W$,则 $i=0, j=j+1$;

(2) 若 k 是奇数,则 $t=2-(k \bmod 4)$, $k=k-t$;

若 $i=0$,则

若 $t=1$,则 $A[j]=A[j] \oplus 0x01$

否则,

若 $j=0$,则 $A[j]=A[j] \oplus 0x03$, $i=i+1$;

否则,

$A[j]=A[j]\oplus 0x01$

$A[j-1]=A[j-1]\oplus 0x80000000$;否则,

若 $t=1$,则 $A[j]=A[j]\oplus(0x01<<i)$;

否则,

$A[j]=A[j]\oplus(0x03<<(i-1))$;

(3) $i=i+1$;

(4) $k=k/2$;

Step 3 返回$(A[j],A[j-1],\cdots,A[0]),i,l=(j-1)\times W+i$

该算法在 k 是偶数时直接将 k 在数值上进行折半处理,相应位取默认值“0”,奇数时需判断产生“1”或者“11”。具体做法是计算如算法 1 中的 t 值,由于 k 是奇数,所以 t 只有两种取值:1 和-1。若 t 为 1,则该位置的 NAF 值为 1;若 $t=-1$,则该位置需向右借 1 位扩展成 11,与此同时还要考虑向右扩展的 3 种情况:

(1) NAF 值的最低字的最低位向右借位扩展:

$$\cdots 1\ 0\ 0\ -1\ 0\ 0\ \boxed{-1}$$

其中,最右的方框中的-1 需要向右扩展,此时最终 NAF 值的位宽会比原位宽多 1,因此算法中对位长单独做了加 1 处理,扩展结果为

$$\cdots 1\ 0\ 0\ 1\ 1\ 0\ \boxed{1}\ \boxed{1}$$

其中,方框中的数值为-1 扩展后的结果。

(2) 字与字间的最低位需向右借位扩展,如表 3 所示。

表 3 字与字间最低位向右借位扩展示例

Table 3 Example of LSB borrow right bit to extend between words

字 W_1	$\cdots$100-101000100-1000000 [-1]
字 W_0	[0] 0001100$\cdots$001100100001001

表 3 中字 W_1 的最低位的-1(方框内)需要向右借位扩展,即向 W_0 的最高位方框内的 0 扩展,因此这步操作涉及到了两个字操作。借位扩展后的结果如表 4 所示。

表 4 字与字间最低位向右借位扩展结果

Table 4 Result of LSB borrow right bit to extend between words

字 W_1	$\cdots$1001110001001100000 [1]
字 W_0	[1] 0001100$\cdots$001100100001001

扩展后位宽只是增加 1,而不是像情况(1)下位长多增加一个 1。

(3) 正常字内向右借位扩展:

$$\cdots 1\ 0\ 0\ 0\ \boxed{-1}\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1$$

字内方框中的-1 需要向右借位扩展,这个情况属于一般情况,正常处理即可,处理结果如下:

$$\cdots 1\ 0\ 0\ 0\ \boxed{1}\ \boxed{1}\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1$$

算法 1 的流程图如图 1 所示。

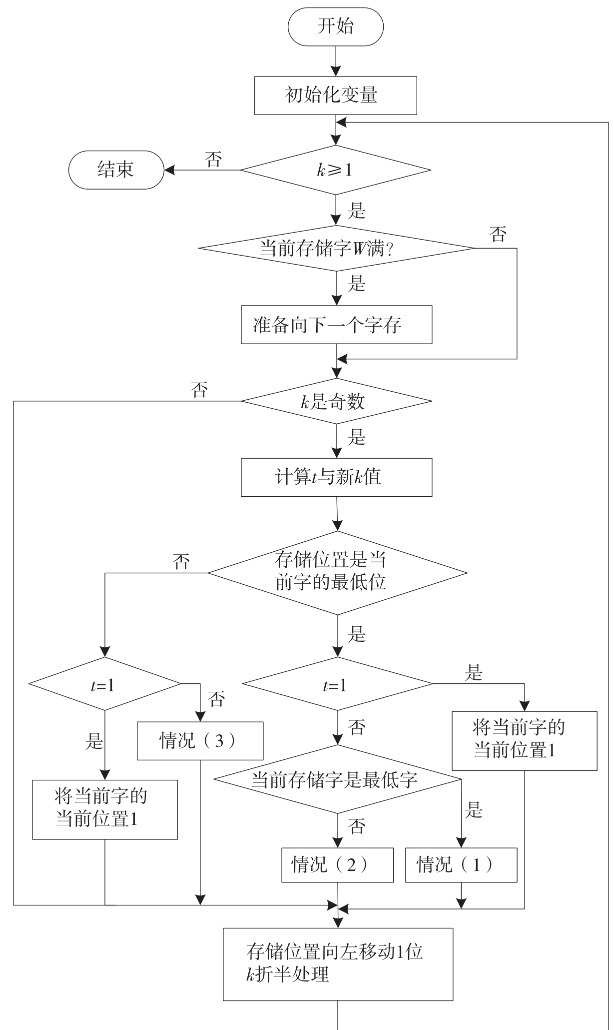

图 1 二进制表示 NAF 算法流程图

Fig. 1 Flow chart of binary representation of NAF algorithm

2.2 计算点乘的二进制 NAF 方法

由于 NAF 算法主要被用于椭圆曲线点乘计算,为了更好地应用算法 1,给出基于算法 1 的点乘算法,具体如算法 2 所示。

文献[1]中定理 3.29 的(v)表明 $NAF(k)$的非零数字平均密度约为 1/3,结合本文中的示例可知

两个非零数值之间至少有一个 0,基于以上规律该算法采用从左向右每两位为一个步长的方式进行探查计算。

算法 2　计算点乘的二进制 NAF 方法

输入:k 的二进制表示序列($A[j], A[j-1], \cdots, A[0]$)及其长度 $l, P \in E(F_q)$

输出:kP

Step 1　用算法 1 计算 $NAF(k)$

Step 2　$Q = \infty$

Step 3　i 从 l 开始直到 1 为止反复执行如下 3 步:

(1) 对($A[j], A[j-1], \cdots, A[0]$)的序列,每次从左到右取两位给 s

(2) $Q = 2Q$

(3) 若 $i = 1$,则

若 $s \neq 0$,则 $Q = Q + P$;

$i = i - 1$;

否则,

① 判断 s 的值

$s = 00$,则 $Q = 2Q$;

$s = 01$,则

若 $i = 2$,则 $Q = 2Q, Q = Q + P$;

否则 s 舍去高位再向后取一位得到新的 s 值,$i = i - 1$,回到第 3.2 步执行;

$s = 10$,则 $Q = Q + P, Q = 2Q$;

$s = 11$,则

若 $i = 2$,则 $Q = Q - P$;

否则,$Q = Q - P, Q = 2Q$;

② $i = i - 2$

Step 4　返回(Q)

kP 的初始值置为∞,它相当于整数域的 0 值;i 为 $NAF(k)$考察位的指示变量,控制算法的起停,它从 $NAF(k)$的位长 l 开始到 1 结束,计算时首先考虑此时 k 考察的位是不是剩下最后一位数值而不够两位,若只剩一位非零数值,这时需要单独进行加点 P 处理,若最后一位是 0 值,则不做点的处理,位置指示变量 i 后移 1 位,返回 Step 3 继续执行,由于指示变量 i 已经为 0,算法结束;若当前考察的 k 不是最后 1 位,则要看每次考察的两位数值是否属于下列 4 种情况:

情况 1　00:直接做一次倍点处理;

情况 2　01:这种情况下有两种可能:

(1) 这个两位是不是最低位,若是最低位表示后边不会出现 1 的可能,即这个 $NAF(k)$ 的最后两位表示为“…01”,此时先做倍点再加点 P 操作,i 向后移动到 0,算法结束;

(2) 若不是最低位则后边有出现 1 的可能(即“…011…”),也有出现 0 的可能(即“…010…”),为了进一步判断具体出现 1 还是 0,需要舍弃当前两位中的高位 0,继续向后取一位与刚刚的两位中的低位“1”重新组成新的两位回到 Step 3 的步骤(2)继续运算考察,当然此时指示变量 i 只移动 1 位;

情况 3　10:对高位的 1 做加点 P 处理,再倍点运算;

情况 4　11:考虑两种情况

(1) 若是 $NAF(k)$的最后两位,即 $i = 2$,则表示原 $NAF(k)$的最右一位为“-1”,11 恰恰就是这个“-1”向右借位扩展得到的两位数值,这里只做减 P 操作,且不需倍点;

(2) 若不是 $NAF(k)$的最后两位,其真实的 $NAF(k)$值为“-10”,对高位需要做减点 P 操作再倍点。

综上,每次考察两位,但是 4 种情况里只对一位进行了倍点,这一位包括情况 1 中的一个“0”,情况 2、3 和 4 中的非零位,那么根据文献[1]中定理 3.29,还需要对每次考察的两位中的“0”值进行倍点,基于此考虑在每次判断两位值之前进行该倍点运算,如算法 Step 3 中的步骤(2)。

算法 2 的流程图如图 2 所示。

3　实验结果及分析

文献[1]中算法 3.30 的存储方式可以有两种:一位一存储方式和只存储非零位方式,前者可采用数组简单的存储数值,后者则需要建立相应的表格,待进行点乘运算时需进行查表确定是加点 P 还是减点 P。由于计算二进制表示的 NAF 过程相同只是在最终结果存储方式有不同,而这个存储方式不会影响其运算时间,因此实验在 Linux 平台对算法 3.30 的一位一存储形式和算法 1 用 C 建模,gcc 编译多组数据进行实验,得到实验结果如表 5 所示。

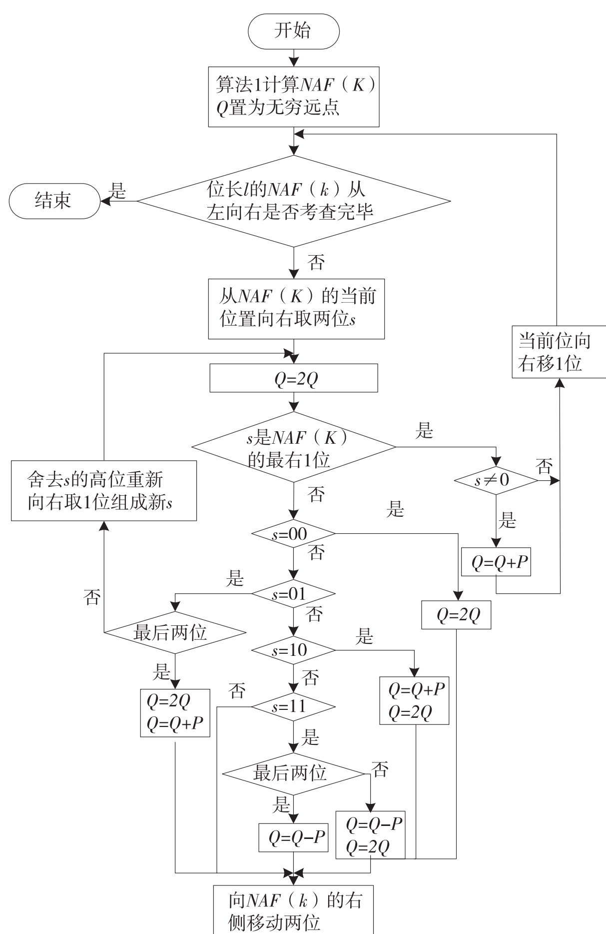

图 2 NAF 二进制表示法的点乘算法流程图

Fig. 2 Flow chart of point multiplication algorithm for NAF binary representation

表 5 两个算法运行时间比较

Table 5 Comparison of running time between two algorithms

数据长度	文献[1]算法 3.30 平均运行时间/μs	算法 1 平均运行时间/μs
163	12	15
233	15	19
283	17	20

从表 5 的结果来看两个算法的平均运行时间相差不多,算法 1 的运行时间较文献[1]算法 3.30 稍长,原因在于文献[1]算法 3.30 与算法 1 的循环次数相同,不同的是当 k 为奇数时算法 1 需要比文献[1]算法 3.30 多做一次字的按位异或运算,根据文献[1]定理 3.29 的(v),算法 1 的按位异或操作有 NAF 长度的 1/3,虽然如此但是两种算法最终返回 NAF 的存储单元个数存在非常大的差异,若文献[1]算法 3.30 需要 l 个字,那么算法 1 则只需要 l/W 个字即可,从空间上节省了 $l-\lceil l/W\rceil$ 个字。若采用只存储非零值且也是一个非零值一个单元存放形式,则根据文献[1]定理 3.29 的(v)需要的存储空间为 $l/3$,相较于算法 1 所需要的存储空间 $\lceil l/W\rceil$ 在存储空间有限的设备上有很大的应用价值。文献[1]算法 3.30 的两种存储方式和算法 1 所占用空间的曲线图如图 3 所示。

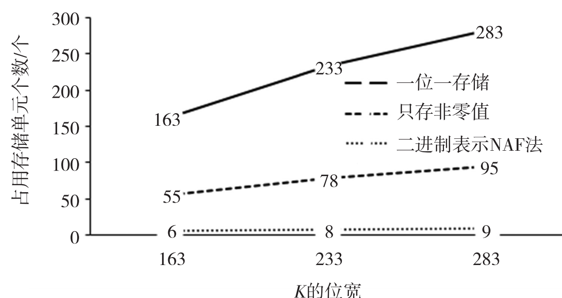

图 3 两种存储方法与算法 1 占用空间比较

Fig. 3 Comparisons of space occupied by two storage methods and algorithm 1

算法 2 与文献[1]算法 3.31 不同的是第 1 步,相较于点加运算的时间复杂度来说其影响是微不足道的。

4 结束语

本文对文献[1]中的 NAF 实现原理进行分析,发现其实现后的数据存储存在很大的优化空间,因此提出了一种新的 NAF 表示方法,用二进制来表示原 NAF 中的 3 个数值:1、0、−1,该方法可以将 NAF 数值存放在存储字中,大大节省了存储空间,而时间上也只受轻微的影响,由于 NAF 算法主要应用在点乘中,因此也给出了用二进制表示 NAF 的点乘算法,该算法的期望运行时间近似于文献[1]中的算法 3.31,但从存储空间上来考察算法 3.31 的空间复杂度大大降低,空间大约节省 96% 以上。

参考文献(References):

[1] DARREL H, ALFRED M, SCOTT V. Guide to Elliptic Curve Cryptography[M]. Springer,2004:92—97

[2] 蒋洪波,尚春雨,冯新宇. NAF 算法的改进[J]. 科学技术与工程,2012,12(19):4663—4665
JIANG H B, SHANG C Y, FENG X Y. Improvement of NAF Algorithm [J]. Science Technology and Engineering. 2012,12(19):4663—4665 (in Chinese)

[3] MIIER B. Improved Techniques for Fast Exponentiation [C]//Information Security and Cryptology 2002 (LNCS 2587) [277], 2003: 298—312

[4] MORAIN F, OLIVOS J. Speeding up the Computations on an Elliptic Curve Using Addition-subtraction Chains [J]. Informatique Theorique et Applications, 1990, 24

(6):531—544

[5] JEROME A. Efficient Arithmetic on Koblitz Curves[J]. Designs, Codes and Cryptography, 2000, 19 (2 - 3): 195—249

[6] 李忠,张永华. 整数的最佳带符号二进制表示的随机生成算法[J]. 计算机科学,2014,41(11):282—283
LI Z, ZHANG Y H. Random Generation Algorithm of Optimal Binary Signed Digit Representation of Integer [J]. Computer Science, 2014, 41 (11): 282—283 (in Chinese)

[7] FIPS 186-2. Digital Signature Standard(DSS). Federal Information Processing Standards Publication 186-2[S]. National Institute of Standards and Technology, 2000

[8] 丁勇. 椭圆曲线快速算法理论[M]. 北京:人民邮电出版社,2012
DING Y. Theory of Fast Algorithms for ECC [M]. Beijing: Posts and Telecommunications Press, 2012 (in Chinese)

[9] 卢开澄,卢华明. 椭圆曲线密码算法导引[M]. 北京:清华大学出版社,2008
LU K C, LU H M. Elliptic Curve Cryptography Guidance [M]. Beijing: Tsinghua University Press, 2008 (in Chinese)

[10] 汪翔,鲍皖苏,吕诗飞. 点乘运算中整数表示方法研究[J]. 微计算机信息. 2006,22(3):240—242
WANG X, BAO W S, LU S F. The Research of the Denotation Methods of Integer in the Point-Multiplication [J]. Microcomputer Information,2006,22(3):240—242 (in Chinese)

[11] GORDON D M. A Survey of Fast Exponentiation Methods [J]. Journal of Algrithms,1998,27(7):129—146

Binary Representation of NAF and Its Algorithm Research

JIANG Hong-bo[1], SUN Yu[2], ZHANG Peng-nan[2], FENG Xin-yu[1], WANG Ming-jie[3]

(1. Heilongjiang Institute of Science and Technology, Harbin 150022, China; 2. NO. 5 Electronics Research Institute of the Ministry of Industry and Information Technology, Guangzhou 510610, China; 3. Harbin Coal Mine Machinery Research Institute, Harbin 150036, China)

Abstract: The fast implementation of elliptic curve cryptography has always been a research hotspot in this field, among which the non-adjacent representation of binary numbers (NAF) is widely used. It is mainly used in point multiplication. The NAF used in this algorithm is composed of digits with symbolic bits, so it usually uses one-by-one storage mode. However, it is great waste on some devices with limited storage resources. In order to save storage resources, a binary representation of NAF is proposed, which can store multiple NAF values according to the word length of the running platform, and greatly improve the utilization of storage resources. On this basis, the algorithm of NAF binary representation and its point multiplication algorithm are given. The experimental results show that the efficiency of this representation has little influence on the efficiency of the original algorithm, especially in point multiplication, but it is outstanding in improving the storage efficiency and saving more than 96% of the storage.

Key words: non-adjacent form; binary representation; point multiplication

责任编辑:罗姗姗

引用本文/Cite this paper:
蒋洪波,孙宇,张鹏南,冯新宇,王明杰. NAF 的二进制表示法及其算法研究[J]. 重庆工商大学学报(自然科学版),2020,37(1):8—13
JIANG H B, SUN Y, ZHANG P N, FENG X Y, WANG M J. Binary Representation of NAF and Its Algorithm Research[J]. Journal of Chongqing Technology and Business University (Natural Science Edition),2020,37(1):8—13