

Fraction-Integer Method (FIM): Calculating Multiplicative Inverse

Sattar J Aboud
Philadelphia University
Computers & Information Systems Dept.
Email: sattar_aboud@yahoo.com

Abdulla M. Abu-Ayyash
Arab Academy for banking and financial Sciences.
Information System Dept.
Email: abuayash@cbj.gov.jo

Abstract

Multiplicative inverse is a crucial operation in cryptographic systems; public key cryptography has given rise to such a need [6], in which we need to generate a related public/private pair of numbers, each of which is the inverse of the other. One of the best methods for calculating the multiplicative inverse is Extended-Euclidean method [5] [3]. In this paper we will propose a new algorithm for calculating the inverse, based on continuous adding of two fraction numbers until an integer is obtained.

1. Introduction

The multiplicative inverse of (e) modulus (n) is an integer (d) $\in \mathbb{Z}_n$ such that $e.d \equiv 1 \pmod n$, d is called the inverse of e and denoted e^{-1} . The study of inverse calculation was a stubborn science due to lack of real improvement, due to [4] the modulus inverse problem is a lot more difficult to solve. However, there were only a couple of methods. One is trivial and lengthy in calculating the inverse, because it is a sequential search. (i.e. start by $d=1$, keep on adding 1 to d until $e.d \equiv 1 \pmod n$). Euclidean (the oldest, yet) the most powerful one, which is based on finding the greater common divisor (gcd) between e and n, such that $\gcd(e,n)=\gcd(e,n \bmod e)$. The algorithm solves x,y such that $e.x+n.y=1$. Stein method [1] [5] which improve Euclidean method by testing for odd and even numbers of e,n, and divide e and/or n by 2 if needed before calculating the inverse. Gordon method [2] is based on using shifts to avoid lengthy multiplication and division. The description, the space requirements and the complexity of each method is discussed in the following sections.

2. Euclidean method

This method is based on the idea that if $n>a$ then $\gcd(a,n) = \gcd(a, n \bmod a)$, also on finding $a.x+y.n=1$ in which x is the multiplicative inverse of a.

2.1 Algorithm of Euclidean method

Input: $a \in \mathbb{Z}_n$ such that $\gcd(a,n)=1$
Output: $a^{-1} \bmod n$, where $a^{-1}=i$ provided that it exists

1. Set $g \leftarrow n, u \leftarrow a, i \leftarrow 0, v \leftarrow 1$.
3. while $u>0$ do the following:
 - 3.1 $q \leftarrow \lfloor g/u \rfloor, t \leftarrow g-qu,$
 - 3.2 $g \leftarrow u, u \leftarrow t, t \leftarrow i -qv$
 - 3.3 $i \leftarrow v.$
 - 3.4 $v \leftarrow t.$
4. if $i < 0$ then $i \leftarrow n+i$.
5. $a^{-1} \leftarrow i$

2.2 Example

Tracing of the algorithm and the variables used is shown in the table below.

Let $a \leftarrow 7, n \leftarrow 60$

| g | u | i | v | q | t |
|----|---|-----|-----|---|-----|
| 60 | 7 | 0 | 1 | 0 | 0 |
| 7 | 4 | 1 | -8 | 8 | -8 |
| 4 | 3 | -8 | 9 | 1 | 9 |
| 3 | 1 | 9 | -17 | 1 | -17 |
| 1 | 0 | -17 | -52 | 3 | -52 |

$$a^{-1} \leftarrow n+i = 60+(-17) = 43$$

2.3 Space and complexity

The method needs around 6 variables, and uses subtraction, multiplication division, and comparison as operations with complexity of $O(\log n)^2$.

3. Stein method

This algorithm was described by [1] and improved by Penk [5] which avoids multiplications. It is based on the observation that $\gcd(x,y)=\gcd(x/2,y)$ if x is even, also $\gcd(x,y)=2\gcd(x/2,y/2)$ if both x, y are even, and $\gcd(x,y) = \gcd((x-y)/2,y)$ if x, y are both odd.

3.1 Algorithm of Stein method

Input: $a \in \mathbb{Z}_n$ such that $\gcd(a, n) = 1$

Output: $a^{-1} \bmod n$, provided that it exists

1. while a and n is even do
 - 1.1 $a \leftarrow \lfloor a/2 \rfloor, n \leftarrow \lfloor n/2 \rfloor$.
2. $u_1 \leftarrow 1, u_2 \leftarrow 0, u_3 \leftarrow a, v_1 \leftarrow n, v_2 \leftarrow 1-a, v_3 \leftarrow n$.
3. if a is odd then $t_1 \leftarrow 0, t_2 \leftarrow -1, t_3 \leftarrow -n$ else $t_1 \leftarrow 1, t_2 \leftarrow 0, t_3 \leftarrow a$
4. repeat
 - 4.1 while t_3 is even do
 - 4.1.1 $t_3 \leftarrow \lfloor t_3/2 \rfloor$.
 - 4.1.2 if t_1 and t_2 is even then $t_1 \leftarrow \lfloor t_1/2 \rfloor, t_2 \leftarrow \lfloor t_2/2 \rfloor$ else $t_1 \leftarrow \lfloor (t_1+n)/2 \rfloor, t_2 \leftarrow \lfloor (t_2-a)/2 \rfloor$.
 - 4.2 if $(t_3 > 0)$ then $u_1 \leftarrow t_1, u_2 \leftarrow t_2, u_3 \leftarrow t_3$ else $v_1 \leftarrow n-t_1, v_2 \leftarrow -(a+t_2), v_3 \leftarrow -t_3$
 - 4.3 $t_1 \leftarrow u_1-v_1, t_2 \leftarrow u_2-v_2, t_3 \leftarrow u_3-v_3$.
 - 4.4 If $(t_1 < 0)$ then $t_1 \leftarrow t_1+n, t_2 \leftarrow t_2-a$.
5. until $t_3=0$.
6. $a^{-1} \leftarrow u_1$.

3.2 Example

Tracing of the algorithm and the variables used is shown in the table below.

Let $a \leftarrow 7, n \leftarrow 60$.

| a | n | u1 | u2 | u3 | v1 | v2 | v3 | t1 | t2 | t3 |
|---|----|----|----|----|----|----|----|-----|----|-----|
| 7 | 60 | 1 | 0 | 7 | 60 | -6 | 60 | 0 | -1 | -60 |
| | | | | | | | | 30 | -4 | -30 |
| | | | | | | | | 15 | -2 | -15 |
| | | | | | 45 | -5 | 15 | | | |
| | | | | | | | | -44 | 5 | -8 |
| | | | | | | | | 16 | -2 | |
| | | | | | | | | 8 | -1 | -4 |
| | | | | | | | | 34 | -4 | -2 |
| | | | | | | | | 17 | -2 | -1 |
| | | | | | 43 | | 1 | | | |
| | | | | | | | | -42 | 5 | 6 |
| | | | | | | | | 18 | -2 | |
| | | | | | | | | 9 | -1 | 3 |
| | | 9 | -1 | 3 | | | | | | |
| | | | | | | | | -34 | 4 | 2 |
| | | | | | | | | 26 | -3 | |
| | | | | | | | | 43 | -5 | 1 |
| | | 43 | -5 | 1 | | | | | | |
| | | | | | | | | 0 | 0 | 0 |

$a^{-1} \leftarrow u_1 = 43$

3.3 Space and complexity

The algorithm needs around 11 variables, and uses addition, subtraction, multiplication, division by 2, and comparison with complexity of $O(\log n)^2$.

4. Gordon Method

This algorithm is based on the observation that (q) at Euclidian algorithm does not need to be the remainder of n/a but it can be any power of 2 up to that limit [2].

4.1 algorithm

Input: $a \in \mathbb{Z}_n$ such that $\gcd(a, n) = 1$

Output : $a^{-1} \bmod n$, provided that it exists

1. $g \leftarrow n, i \leftarrow 0, v \leftarrow 1, u \leftarrow a$.
2. repeat
 - 2.1 $s \leftarrow -1, p \leftarrow 0$.
 - 2.2 If $u > g$ then
 - 2.2.1 $t \leftarrow 0$
 - 2.3 else
 - 2.3.1 $p \leftarrow 1, t \leftarrow u$.
 - 2.3.2 while $(t \leq g)$ do
 - 2.3.2.1 $s \leftarrow s+1$.
 - 2.3.2.2 $t \leftarrow$ left shift t by 1.
 - 2.3.3 $t \leftarrow$ right shift t by 1.
 - 2.4 $t \leftarrow g-t, g \leftarrow u, u \leftarrow t, t \leftarrow i, i \leftarrow v$.
 - 2.5 if $p=1$ then
 - 2.5.1 $v \leftarrow$ left shift v by s .
 - 2.5.2 $t \leftarrow t-v$.
 - 2.6 $v \leftarrow t$.
3. until $u=0$ or $u=g$.
4. if $i < 0$ then $i \leftarrow n+i$.
5. $a^{-1} \leftarrow i$.

4.2 Example

Tracing of the algorithm and the variables used is shown in the table below.

Let $a \leftarrow 7, n \leftarrow 60$.

| g | u | i | v | s | p | t |
|----|---|----|----|----|---|-----|
| 60 | 7 | 0 | 1 | 0 | 1 | 14 |
| | | | | 1 | | 28 |
| | | | | 2 | | 58 |
| | | | | 3 | | 112 |
| | | | | | | 56 |
| 7 | 4 | | | | | 4 |
| | | 1 | | | | 0 |
| | | | 8 | | | -8 |
| | | | -8 | | | |
| | | | | -1 | 0 | |
| | | | | | 1 | 4 |
| | | | | 0 | | 8 |
| | | | | | | 4 |
| 4 | 3 | | | | | 3 |
| | | -8 | | | | 1 |
| | | | | | | 9 |
| | | | 9 | | | |
| | | | | -1 | 0 | |

| | | | | | | |
|---|---|-----|-----|----|---|-----|
| | | | | | 1 | 3 |
| | | | | 0 | | 6 |
| | | | | | | 3 |
| 3 | 1 | | | | | 1 |
| | | 9 | | | | -8 |
| | | | | | | -17 |
| | | | -17 | | | |
| | | | | -1 | 0 | |
| | | | | | 1 | 1 |
| | | | | 0 | | 2 |
| | | | | 1 | | 4 |
| | | | | | | 2 |
| 1 | | | | | | 1 |
| | | -17 | | | | 9 |
| | | | -34 | | | 43 |
| | | | 43 | | | |

$$a^{-1} \leftarrow -60 - 17 = 43$$

4.3 Space and complexity

The algorithm needs around 7 variables, and uses addition, subtraction, comparison and shifts with complexity of $O(\log n)$

5. Fraction-Integer Method

The idea behind the proposed method is very simple, including the division by 1, n by e and keep on adding n/e to 1/e until an integer obtained.

5.1 Algorithm of Fraction-Integer method

Input: $a \in \mathbb{Z}_n$ such that $\gcd(a, n) = 1$
Output: $a^{-1} \bmod n$, provided that it exists

1. Let $d \leftarrow 1/a$
2. Let $def \leftarrow n/a$
3. repeat
 - 3.1 $d \leftarrow d + def$.
4. until d is integer
5. $a^{-1} \leftarrow d$.

5.2 Example

Tracing of the algorithm and the variables used is shown in the table below.

Let $a \leftarrow 7$, $n \leftarrow 60$.

| D | def |
|---------|--------|
| 0.1429 | 8.5714 |
| 8.7143 | |
| 17.2857 | |
| 25.8571 | |
| 34.4286 | |
| 43.0000 | |

$$a^{-1} \leftarrow d.$$

5.3 Space and complexity

The algorithm needs only 2 variables, and uses addition and comparison with complexity of $O(\text{size floating point registers})$

6. Proof of Fraction-Integer Method (FIM)

In order to prove the algorithm, we need to prove that the algorithm will give integer number only when d is the inverse of e.

As we know that if d is the inverse of e then

- 1- Both e, d are positive integer numbers between $[1, n]$ (1)
- 2- $\gcd(e, n) = 1$ (2)
- 3- $e * d \equiv 1 \bmod n$, i.e. $e * d = 1 + k * n$ (for $k \in \mathbb{Z}$), (3)

so

$$d = (1 + k * n) / e = 1/e + k * n/e \dots\dots (4)$$

From the algorithm we see that

$$d = 1/e + (def + def + \dots + def) \text{ i times until d is integer.}$$

$$d = 1/e + i * def = 1/e + i * n/e \dots\dots (5)$$

From that we know that the algorithm above is correct for $i=k$, but if this is the case we need to prove that (5) will give a none integer for all values of $i < k$, and the only integer value is when $i=k$, so we know d is an integer so $(1+k*n)/e$ is also an integer for an integer value of k. Assume that this is true for some value k (by definition (3,4))

Then we need to proof that $(1+i*n)/e$ is never an integer for all values of i between $[1, k-1]$. Assume that there is another value of i , $1 < i < k$ such that $d = (1+i*n)/e$ is also an integer, i.e.

$$i=k-1 \dots\dots\dots (6)$$

Then $d = (1 + (k-1)*n)/e$ will be integer. So

$$\begin{aligned} d &= (1 + k*n - n) / e \\ &= (1 + k*n) / e - n/e \\ &= 1/e + k*n/e - n/e \end{aligned}$$

But by definition (1,3,4) we know that $1/e + k*n/e$ is integer , also that $\gcd(e, n)$ should be 1 (2) so if there is no greater common divisor between e and n except 1, that mean n/e is a non integer value. So subtracting a non integer value form an integer value will yield d is not an integer. Which contradicts our assumption (that d is an integer)..... (6).

Now assume that there exist an $i=k-q$ such that d is an integer for q between $[1, k-1]$. Then $d=(1+(k-q)*n)/e=1/e + k*n/e - q*n/e$, and if this to be integer then $q*n/e$ must be integer, but since $\gcd(e,n)=1$ then q must be a multiple of e so $d=1/e+k*n/e-x*n$ (5)

This will lead to d being a negative number $d<0$ but from definition we know that both e,d must be positive (1) so there is no values for x that satisfy the definition. So the only value for q that satisfy the conditions is when $q=0$ and that $i=k$ (done).

7. Problem of FIM method

We have proved that FIM algorithm is correct, but the question is that is it implementable ? i.e. the algorithm will terminate giving the correct answer when implemented using the computer programming languages ?

Let d_m be the mathematical value of d where $d=d_m$.

Let d_c be the calculated value of d in the computer memory and registers.

Let ζ be the error in calculating, between the mathematical value and the computer value (round off error). So

$$\begin{aligned} d_m &= (1_m + k_m * n_m) / e_m \text{ so} \\ &= 1_m / e_m + k_m * n_m / e_m \\ &= (1/e)_m + (k*n/e)_m \end{aligned}$$

But we know that the calculated value of fractions is never exactly as the mathematical value for big values of e that when used to divide 1 and n will give a cyclic fraction number, so $(1/e)_m = (1/e)_c + \zeta_1$ and $(n/e)_m = (n/e)_c + \zeta_2$ where $\zeta_1 \ll (1/e)_c$ and $\zeta_2 \ll (n/e)_c$, and $d_c = (1/e)_c + (k * n / e)_c + \zeta_1 + k * \zeta_2$ such errors will yield that either $d_m \leq d_c$ or $d_m \geq d_c$, $d_m = d_c$ if and only if $\zeta_1 + k * \zeta_2 = 0$ i.e. $(1/e)_m = (1/e)_c$ and $(n/e)_m = (n/e)_c$. We know that the error ζ_1, ζ_2 is small, but multiplying ζ_2 with k will produce big value off error, so as k increase the error also will increase, so the best approach is to use small values for e .

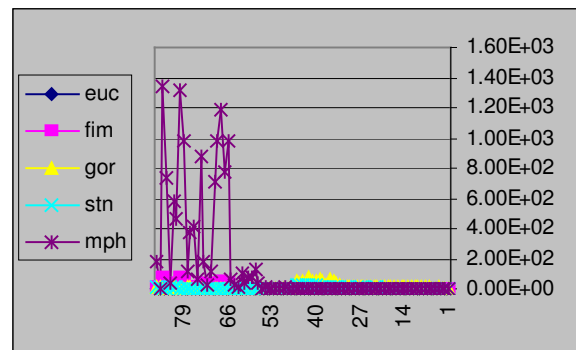
8. Timing

The MATLAB 5.1 is used to compare the proposed FIM algorithm with some of known algorithms [Extended Euclid, Stein, and Gordon methods] and is shown in figures 1-4. We have implemented the algorithm for different numbers from one digit to 6 digits for e numbers and the result are shown in the figures below. We noticed that the time for Extended Euclid algorithm is approximately irrelevant to e or n , but other algorithms is affected by e and n . The proposed FIM algorithm outperform the other methods for small number of e and irrelevant to n . As we can see that FIM algorithm is based only on addition which is the fastest operation, and that is

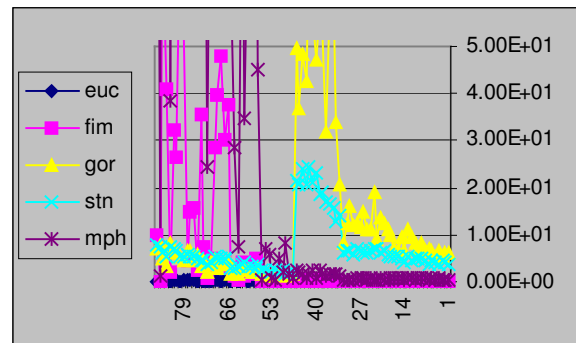
why it outperform the other methods except Euclid for big numbers of e .

9. Conclusion

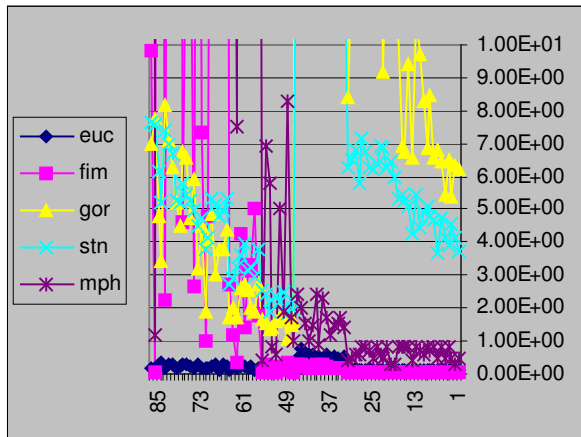
For security reasons, cryptography recommends smaller values for public keys and bigger values for private keys [3]. The suggested algorithm needs small values for public keys (lower value of e) and big values for private key, which is fully compatible with the preferred cryptographic algorithm. The method is simple, fast and needs less storage, and its complexity is also less.



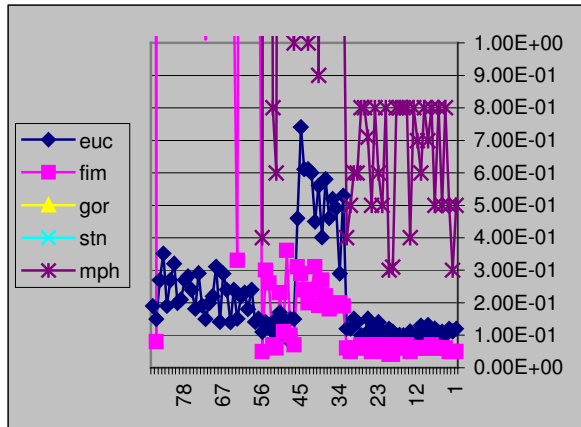
Figure(1) comparison between methods big scale of time



Figure(2) comparison between methods 50 sec scale of time



Figure(3) comparison between methods 10 sec scale of time



Figure(4) comparison between methods 1 sec scale of time

Note: FIM outperform over other methods for small values of e.

11. References

- [1] Stein, J. (1967) J. Comp. Phys, 1, p397-405.
- [2] J. Gordon, "Fast Multiplicative inverse in modular arithmetic", cryptography and coding, clarendon press oxford, 1989.p269-279.
- [3] Menezes A. et al, "Handbook of applied cryptography", 1996 p67, p71.
- [4] Bruce Schneier "applied Cryptography", Second Edition, John Wiley and sons, 1996.p246.
- [5] Knuth, D. E., "The art of computer programming - Vol. 2 semi numerical algorithms', 2nd Ed., Addison-Wesley, 1981,pp 319,321,339, 599.
- [6] Rivest R., Shamir A., Adlemen L., "A method for obtaining digital signatures and public key cryptosystems", Comms. ACM, 1978, 21,2, 120-126.