

Aryabhata Remainder Theorem: Relevance to public-key crypto algorithms

T.R.N. Rao¹

楊中皇²
 Chung-Huang Yang

Abstract Public-key crypto-algorithms are widely employed for authentication, signatures, secret-key generation and access control. The new range of public-key sizes for *RSA* and *DSA* has gone up to 1024 bits and beyond. Elliptic-curve key range is from 162 bits to 256 bits. Many varied software and hardware algorithms are being developed for implementation for smart-card crypto-coprocessors and for public-key infrastructure. We begin with an algorithm from *Aryabhataiya*, for solving the indeterminate equation $a \cdot x + c = b \cdot y$ of degree one (also known as Diophantine equation) and its extension to solve the system of two residues $X \bmod m_i = X_i$ (for $i=1, 2$). This contribution known as Aryabhata Algorithm (AA) is very profound in the sense that the problem of two congruences was solved with just one modular inverse operation and a modular reduction to a smaller modulus than the compound modulus. We extend AA to any set of t residues and is stated as Aryabhata Remainder Theorem (ART) and an iterative algorithm is given to solve for t moduli m_i ($i=1, 2, \dots, t$). The ART, which has much in common with Extended Euclidean Algorithm (EEA), Chinese Remainder Theorem (CRT) and Garner's algorithm (GA), is shown to have a complexity comparable or better than CRT and GA

Keyword Arithmetic, Diophantine, remainder, residue, inverse

1 Introduction

We begin with an algorithm of Aryabhata[#] found in the text *Aryabhataiya* [1, 2, 3, 12], which solves the linear indeterminate equation, $a \cdot x + c = b \cdot y$, for positive integers a, b and c (sometimes called Diophantine equation). This algorithm also solves for X , given the pair of residues $X \bmod m_i = x_i$ (for $i = 1, 2$). There is some underlying principle of simplicity in this solution, which has been found to be applicable to solving the general case of n residues in an iterative manner and requiring as few inverse operations as any we know of today and also without the necessity for a final modular reduction operation. This leads us to state these as Aryabhata Remainder Theorem (ART) and the ART algorithm presented here is comparable and in some ways more efficient than the CRT algorithm of Gauss [4], the original Garner's Algorithm [5] (and) the later version of GA given in [6,7].

1.1 Improved Aryabhata Algorithm (IAA)

Aryabhata Algorithm (AA)

In the field of pure mathematics, one of the most significant contributions of Aryabhata is his solution to the indeterminate equation $a \cdot x + c = b \cdot y$. We copy here the example and discussion given by Pearce [8] from *Aryabhataiya* [2, 3, 12]

Example 1: $137 \cdot x + 10 = 60 \cdot y$ (1)

60)137(2 (60 divides into 137 twice with remainder 17)

$$\begin{array}{r}
 137 \\
 \underline{120} \\
 17 \\
 17)60(3 \\
 \underline{51} \\
 9 \\
 9)17(1 \\
 \underline{9} \\
 8 \\
 8)8(1 \\
 \underline{8} \\
 0
 \end{array}$$

The column of quotients known as *Valli* (Vertical line)

¹Center for Advanced Computer Studies, University of Louisiana at Lafayette, P.O. Box 44330, Lafayette, Louisiana, 70504-4330, USA, trao@cacs.louisiana.edu

²国立高雄師範大學, National Kaohsiung Normal University, 116, Heping First Road, Lingya District, Kaohsiung 802, TAIWAN, http://crypto.nknu.edu.tw

[#]Pearce [8] states: "we can accurately claim that Aryabhata was born in 476 A.D., as he writes that he was 23 years old when he wrote his most significant mathematical work *Aryabhataiya* in 499 A.D...."

is constructed

2
3
1
1

This method was called *Kuttaka*, which literally means ‘pulverizer’, on account of the process of continued division that is carried out to obtain successively smaller remainders.

This process in traditional textbook format of n equations is as follows:

$$\left. \begin{aligned} a &= b \cdot q_1 + r_1 \\ b &= r_1 \cdot q_2 + r_2 \\ &\vdots \\ r_{n-2} &= r_{n-1} \cdot q_n + r_n \end{aligned} \right\} \quad (2)$$

The number of quotients, omitting the first one is 3, which is odd. Hence we choose a multiplier k such that on multiplication by the last residue, 1 (in dashed rectangle), and subtracting 10 from the product the result is divisible by the penultimate remainder, 8 (in dashed circle). If the number of quotients after omitting the first one, is even then adding 10 is required instead of subtracting.

We have $1 \cdot 18 - 10 = 1 \cdot 8$. Then we can form the following table:

	2	2	2	2	297
	3	3	3	130	130
	1	1	37	37	
	1	19	19		
$k =$	18	18			
	1				

This can be explained as follows:
The number 18, and the number above it in the first column, multiplied and added to the number below it, gives the last but one number in the second column. Thus, $18 \cdot 1 + 1 = 19$. The same process is applied to the second column, giving the third column that is $19 \cdot 1 + 18 = 37$. Similarly $37 \cdot 3 + 19 = 130$, $130 \cdot 2 + 37 = 297$. Then $x = 130$, $y = 297$ are solutions of the given equation. Noting that $297 \pmod{137} = 23$ and $130 \pmod{60} = 10$, we get $x = 10$ and $y = 23$ as simple solutions. Thus we have $137 \cdot 10 + 10 = 60 \cdot 23$ as a solution for Equation (1).

1.2 Improved Aryabhata Algorithm (IAA)

We show how to simplify considerably the above procedure. First replace c with 1 in the Equation (1) to get:

Example 2: $137 \cdot x + 1 = 60 \cdot y$ (3)

Using *Kuttaka* we obtain q_i as before. Form a table with the first column as the iteration variable i , and the *Valli* of q_i as the second column. The third column of S_i is obtained from bottom up in a similar manner as in the Example 1. We start with $S_5 = 1$, $S_4 = q_4$ as initial values and use the recursion formula:

$$S_i = q_i \cdot S_{i+1} + S_{i+2} \quad (4)$$

i	q_i	S_i
1	2	16
2	3	7
3	1	2
4	1	1
5		1

We compute the S_i 's from the bottom up

$$\begin{aligned} S_3 &= q_3 \cdot S_4 + S_5 = 1 \cdot 1 + 1 = 2 \\ S_2 &= q_2 \cdot S_3 + S_4 = 3 \cdot 2 + 1 = 7 \\ S_1 &= q_1 \cdot S_2 + S_3 = 2 \cdot 7 + 2 = 16 \end{aligned}$$

We note the number of quotients $n = 4$, and $\gcd(137, 60) = d = 1$. We give the answer to Equation (4) as $137 \cdot S_2 + (-1)^n d = 60 \cdot S_1$. Thus, we have $137 \cdot 7 + 1 = 60 \cdot 16$.

To solve Equation (1), we multiply the above by 10 to get $137 \cdot 70 + 10 = 60 \cdot 160$. By taking out $137 \cdot 60$ from both sides, we get the simple solution $137 \cdot 10 + 10 = 60 \cdot 23$.

The relevance of the solution $137 \cdot 7 + 1 = 60 \cdot 16$ for us is that $137^{-1} \pmod{60} = -7 \pmod{60} = 53$ and $60^{-1} \pmod{137} = 16$. Thus we get both inverses, $a^{-1} \pmod{b}$ and $b^{-1} \pmod{a}$ by this method. Also these lead us to the solution to the problem of two residues as shown in Section 2.

The theory behind the results of the above two examples can be put in the form of two lemmas as follows (proofs are quite simple and are given in [13]).

Lemma 1: Let $a, b, c, q_i, r_i, S_i, d$ and n are as defined in the previous examples. The solution (if it exists) to $a \cdot x + d = b \cdot y$ is given by:

$$\begin{aligned} x &= [(-1)^n \cdot S_2 \cdot c/d] \pmod{b} \\ y &= [(-1)^n \cdot S_1 \cdot c/d] \pmod{a} \end{aligned}$$

Lemma 2: For $a \cdot x + d = b \cdot y$, IAA yields a solution in the proper range, i.e. $0 < S_1 < a$, and $0 < S_2 < b$.

2 The Problem of Two Residues

Consider $X \pmod{60} = 0$ and $X \pmod{137} = 10$. Clearly $X = 60y$ for some y and also $X = 137x + 10$ for some x . That means solving $137 \cdot x + 10 = 60 \cdot y$,

which we did in the previous section. Thus $X = 60 \cdot 23 = 1380$. Let us now modify the problem slightly by adding a 5 to both of the residues. Then we have

$$X \bmod 60 = 5 \text{ and } X \bmod 137 = 15.$$

The answer here is just to add 5 to the previous solution and we get $X = 1385$. This method is given as residual pulveriser (*Saagra-kuttaakaara* in Aryabhatia [3]). Here is an important underlying principle in Aryabhata's solution to the problem of two residues. That is, to solve the problem of two residues first solving the Diophantine equation $a \cdot x + c = b \cdot y$ and then adding a constant to the result. Solving the equation amounts to finding the modular inverse of $b \bmod a$ and then a modular multiplication with $c \bmod a$. This is a profound and significant contribution for which Aryabhata should be recognized by the cryptology community. Its extension for the t -moduli we present here will be also of great importance due to the PKCS #1 v2.1 of RSA cryptographic standard [10], which discusses modulus n of 2048 bits, a composite of four primes each of 512 bits. In this context, every contribution in residue operations, and number conversions will become important for now and the future.

This was called Aryabhata Algorithm (AA) by Kak [1]. That paper also contained a detailed historical presentation on the system of multiple residues in India and the work of Sun Tzu and others in China. He also discusses how the Aryabhata algorithm was used to solve problems in Astronomy in India. Here we develop the solution as a formal theorem and call it Aryabhata Remainder Theorem (ART) as a tribute to perhaps the greatest mathematician and astronomer of the classical period (5th century to 12th century A. D.)

Aryabhata Remainder Theorem (ART): Let m_1 and m_2 be relatively prime moduli and $M = m_1 m_2$.

$$\text{Given } X \bmod m_1 = x_1, \quad X \bmod m_2 = x_2,$$

X has a unique solution in Z_M given by:

$$\begin{aligned} X &= \text{ART}(x_1, x_2; m_1, m_2; M) \\ &= \text{ART}(0, c; m_1, m_2; M) + x_1 \end{aligned}$$

$$\text{where } c = (x_2 - x_1) \bmod m_2$$

$$= A + x_1, \quad \text{where } A = m_1 [(c \cdot m_1^{-1}) \bmod m_2].$$

Proof: First we show that $X = A + x_1 \in Z_M$. Since $A = m_1 \cdot b$ for some $b \in Z_{m_2}$, A must be less than or equal to $m_1(m_2-1)$. Since $x_1 < m_1$, $A + x_1$ must be less than $M = m_1 m_2$ and therefore $X \in Z_M$. Now consider $(A + x_1) \bmod m_1$. Since A is a multiple of m_1 , we have $(A + x_1) \bmod m_1 = x_1$. Since $A \bmod m_2 = c$ due to the cancellation of the terms m_1 and m_1^{-1} , we have $(A + x_1) \bmod m_2 = c + x_1 = x_2$. Thus $A + x_1 = X$ satisfies the two congruences as

required and is a solution in Z_M . It is easy to show that $A + x_1$ is a unique solution in Z_M . If $Y \in Z_M$ is another solution, then $(X - Y) \bmod m_i = 0$, for $i = 1, 2$ and $(X - Y) \bmod M = 0$. Thus $X = Y$.

A formal extension of ART to three moduli is also given in *Aryabhatiya* and is called *tricedaagra* (the number to three divisors/residues [3]), *triceda* literally means 3 divisions and *agra* means first. The ART theorem for n moduli is straight forward and is given Section 5. Here we illustrate it by an example.

Example 3: Let $X \bmod 3 = x_1 = 1$, $X \bmod 4 = x_2 = 3$, and $X \bmod 5 = x_3 = 3$.

$$\text{Then } X = \text{ART}(1, 3, 3; 3, 4, 5; 60)$$

Step 1

$$\begin{aligned} X' &= X \bmod 12 = \text{ART}(1, 3; 3, 4; 12) \\ &= \text{ART}(0, 2; 3, 4; 12) + 1 \\ &= 3 [(2 \cdot 3^{-1}) \bmod 4] + 1 \\ &= 3 \cdot 2 + 1 = 7 \end{aligned}$$

Step 2

$$\begin{aligned} X &= \text{ART}(7, 3; 12, 5; 60) \\ &= \text{ART}(0, (3-7) \bmod 5; 12, 5; 60) + 7 \\ &= \text{ART}(0, 1; 12, 5; 60) + 7 \\ &= 12 [(1 \cdot 12^{-1}) \bmod 5] + 7 \\ &= 12 \cdot 3 + 7 = 43 \end{aligned}$$

3 Multiplicative Inverse

Given positive pairwise prime integers a and b , it is very often necessary to find $a^{-1} \bmod b$. That is, to find $x \in Z_b$ such that $a \cdot x \bmod b = 1$. In RSA, the private key d is generated by finding the inverse of public-key $e \bmod \phi(n)$, where $\phi(n) = (p-1)(q-1)$. The Extended Euclidean Algorithm (EEA) [6, 7] given below obtains $a \cdot x + b \cdot y = 1$, for given a and b . Finding the multiplicative inverse is illustrated by the Example (4) given below:

Extended Euclidean Algorithm: The extended Euclidean algorithm is available in most texts [6,7]. The simpler version of Euclidean algorithm from [11] will illustrate the principle as well.

Example 4: Let $a = 137$ and $b = 60$.

i	r_i	q_i	x_i	y_i
-1	137		1	0
0	60	-	0	1
1	17	2	1	-2
2	9	3	-3	7
3	8	1	4	-9
4	1	1	-7	16
5	0			

$$q_i \leftarrow \text{quotient}(r_{i-2}/r_{i-1}), \quad r_i \leftarrow r_{i-2} \bmod r_{i-1}$$

$$x_i \leftarrow x_{i-2} - q_i \cdot x_{i-1}, \quad y_i \leftarrow y_{i-2} - q_i \cdot y_{i-1}$$

From the table above we have $x = -7 \pmod{60} = 53$ and $y = 16$. Therefore $137^{-1} \pmod{60} = -7 \pmod{60} = 53$ and $60^{-1} \pmod{137} = 16$.

EEA requires series of successive division steps as in GCD algorithm, while calculating x_i and y_i iteratively and ultimately to obtain the final values. This procedure requires n divisions, $2n$ multiplications, and $2n$ subtractions, where n is the number of iterations; However IAA requires n divisions and $n-2$ multiplications and $n-2$ additions to find s_2 , the inverse of $a \pmod{b}$. While EEA derives the values x_i and y_i in a forward direction as q_i are generated, IAA will have to generate all q_i 's and then apply the iterations in a reverse manner. This requires storing q_i 's and is indeed an undesirable feature. Thus EEA algorithm is superior in that sense. Also if one needs only one inverse (i.e., $a^{-1} \pmod{b}$) y_i column is not required and in that case, just n multiplication and subtraction steps are needed. Knuth [6] obtains n , the average number of divisions required for GCD for given x and a random $y < x$ by the formula

$$n \approx 1.94 \log_{10} x.$$

For a 100 digit decimal number a , and a randomly chosen $b < a$, the average number of division steps will be about 194.

Multiplicative Inverse Algorithm

EEA can be improved to perform better if only one inverse is required. For instance if

$a^{-1} \pmod{b}$ is required for $a > b$, We may just as well begin with $a \pmod{b} = c$ and find $c^{-1} \pmod{b}$. In that case, the x_i computation will be of one less step. Further if the initial values are set appropriately the inverse can be obtained in $n-2$ forward steps (each step: one multiplication and one addition) the same number of steps as in IAA (Section 1.2). We illustrate this by the same example as before and by a table given below.

Example 5: Find: $137^{-1} \pmod{60}$ ($a = 137$ and $b = 60$)

We start with $r_0 = b = 60$, $r_1 = a \pmod{60} = 17$ and $x_1 = 1$. The iterations begin from $i = 2$ with the normal division process: $q_i \leftarrow \text{quotient}(r_{i-2}/r_{i-1})$, $r_i \leftarrow r_{i-2} \pmod{r_{i-1}}$ and $x_2 = q_2$.

The iteration proceeds: $x_i \leftarrow x_{i-1} \cdot q_i + x_{i-2}$ (for $i > 2$).

i	r_i	q_i	x_i
0	60	-	-
1	17	-	1
2	9	3	3
3	8	1	4
4	1	1	7
5	0		

From the above we observe the following:

$$a \cdot x_i (-1)^{i-1} \pmod{b} = r_i \text{ for } i \geq 1, \quad 137 \cdot 7(-1)^{4-1} \pmod{60} = 1,$$

$$X = 137^{-1} \pmod{60} = 60 - 7 = 53.$$

We can now state the following

Lemma 3: Let a, b, r_i, q_i and x_i be defined as above. Then $a^{-1} \pmod{b}$ exists iff $x_n = 1$, (for some $n > 1$) and is given by

$$a^{-1} \pmod{b} = x_n (-1)^{n-1}$$

Proof: First, we need to prove that $a \cdot x_i (-1)^{i-1} \pmod{b} = r_i$ holds for $i \geq 1$. For $i = 1$, we have $x_1 = 1$ and $a \cdot x_1 (-1)^{1-1} \pmod{b} = r_1$. For $i = 2$, we have the division equation $r_2 = r_0 - q_2 \cdot r_1 = r_0 - x_2 \cdot r_1$. Taking \pmod{b} on both sides, we get $(-x_2) \cdot r_1 \pmod{b} = r_2$, which is same as $(-x_2) a \pmod{b} = r_2$. For $i = 3$, we start with $r_3 = r_1 - q_3 \cdot r_2 = r_1 \cdot x_1 - q_3 (r_0 - x_2 \cdot r_1) = r_1 (x_2 \cdot q_3 + x_1) - q_3 \cdot r_0 = r_1 \cdot x_3 - q_3 \cdot r_0$. Taking \pmod{b} on both sides, we have $r_1 \cdot x_3 \pmod{b} = r_3$ and $a \cdot x_3 \pmod{b} = r_3$. Continuing this process, we obtain $a \cdot x_n (-1)^{n-1} \pmod{b} = r_n = 1$ and $a^{-1} \pmod{b} = x_n (-1)^{n-1}$.

Algorithm for $a^{-1} \pmod{b}$:

Step 1. $r_0 \leftarrow b$

$r_1 \leftarrow a \pmod{b}$

if $r_1 = 0$, then goto Step 4

else $x_1 \leftarrow 1$

$i \leftarrow 2$

Step 2. $q_i \leftarrow \text{quotient}(r_{i-2}/r_{i-1})$

$r_i \leftarrow r_{i-2} \pmod{r_{i-1}}$

If $r_i = 0$, then go to step 3

else if $i = 2$, then $x_i \leftarrow q_i$

else $x_i \leftarrow x_{i-1} \cdot q_i + x_{i-2}$

$i \leftarrow i + 1$

go to Step 2

Step 3. If $r_{i-1} = 1$, then if i is even,

then return (x_i)

else return $(b - x_i)$

Step 4. print "Inverse does not exist"

4 Chinese Remainder Theorem (CRT)

Let $X = CRT(v_1, v_2, \dots, v_t; m_1, m_2, \dots, m_t; M = \prod_{i=1}^t m_i)$ for $(m_i, m_j) = 1$, for all $i \neq j$. Then X is given by

$$X = [\sum_{i=1}^t v_i (M/m_i) y_i] \pmod{M}, \text{ where } y_i = (M/m_i)^{-1} \pmod{m_i}$$

Example 5: $X = CRT(2, 1, 3, 8; 5, 7, 11, 13; 5005)$

$$y_1 = (5005/5)^{-1} \pmod{5} = (1001)^{-1} \pmod{5} = 1$$

$$y_2 = (5005/7)^{-1} \pmod{7} = (715)^{-1} \pmod{7} = 1$$

$$y_3 = (5005/11)^{-1} \pmod{11} = (455)^{-1} \pmod{11} = 3$$

$$y_4 = (5005 / 13)^{-1} \text{ mod } 13 = (385)^{-1} \text{ mod } 13 = 5$$

$$X = 2 \cdot 1001 \cdot 1 + 1 \cdot 715 \cdot 1 + 3 \cdot 455 \cdot 3 + 8 \cdot 38 \cdot 5 \\ = (2002 + 715 + 4095 + 385) \text{ mod } 5005 = 2192$$

Comments: CRT requires t inverse operations and a reduction operation modulo M . As explained in [7] the number of bit operations $O(k^2 \cdot t^2) = O(n^2)$, where n is the combined number of bits in modular representation of $v(x)$.

Garner's Algorithm (GA)

Garner [5] deduced an algorithm to convert residue code of a number $X = (v_1, v_2, \dots, v_t)$ with respect to pairwise relatively prime modulo m_1, m_2, \dots, m_t to a mixed radix number with weight 1, $m_b, m_{t-1}m_t$, and so on upto the last one $m_2m_3\dots m_t$. Then its radix equivalent can be easily computed using those weights. As example he chose (1, 2, 0, 4) for moduli set (2, 3, 5, 7) and converted to mixed radix form of (0, 2, 3, 4) whose weights are (105, 35, 7, 1) respectively. Then $X = (1, 2, 0, 4)$ represented $0 \cdot 105 + 2 \cdot 35 + 3 \cdot 7 + 4 = 95$. A refined version of Garner's algorithm has been given in [7] as follows:

INPUT : a positive integer $M = \prod_{i=1}^t m_i$, with $\text{gcd}(m_i, m_j) = 1$ for all $i \neq j$, and a modular representation $v(x) = (v_1, v_2, \dots, v_t)$ of x for the m_i .

OUTPUT: the integer x in radix b representation.

1. For i from 2 to t do the following:
 - $C_i \leftarrow 1$.
 - For j from 1 to $(i-1)$ do the following:
 - $u \leftarrow m_j^{-1} \text{ mod } m_i$
 - $C_i \leftarrow u \cdot C_i \text{ mod } m_i$
2. $u \leftarrow v_1, x \leftarrow u$
3. For i from 2 to t do the following :
 - $u \leftarrow (v_i - x) \cdot C_i \text{ mod } m_i$,
 - $x \leftarrow x + u \cdot \prod_{j=1}^{i-1} m_j$
4. Return (x) .

x returned by Algorithm (GA) satisfies $0 \leq x < M, x \equiv v_i \pmod{m_i}, 1 \leq i \leq t$.

Example 6: (Garner's algorithm) [7]: Let $m_1 = 5, m_2 = 7, m_3 = 11, m_4 = 13, M = \prod_{i=1}^4 m_i = 5005$, and $v(x) = (2, 1, 3, 8)$. The constants C_i computed are $C_2 = 3, C_3 = 6, C_4 = 5$. The values if (i, u, x) computed in step 3 of algorithm are (1, 2, 2), (2, 4, 22), (3, 7, 267) and (4, 5, 2192). Hence, the modular representation $v(x) = (2, 1, 3, 8)$ corresponds to the integer $X = 2192$.

Menezes et. al. [7] provide a discussion on

computational efficiency of GA as follows:

"If Garner's algorithm is used repeatedly with the same modulus M and the same factors of M , then step 1 can be considered as a precomputation, requiring the storage of $t-1$ numbers. The classical algorithm for the CRT typically requires a modular reduction with modulus M , whereas Garner's algorithm does not. Suppose M is a kt -bit integer and each m_i is a k -bit integer. A modular reduction by M takes $O((kt)^2)$ bit operations. Whereas a modular reduction by m_i takes $O(k^2)$ bit operations. Since Garner's algorithm only does modular reduction with $m_i, 2 \leq i \leq t$, it takes $O(t \cdot k^2)$ bit operations in total for the reduction phase, and is thus more efficient."

However, GA requires $t(t-1)/2$ inverse $\text{mod } m_i$ operations. Since inverse requires $O(k^2)$ bit operations the complexity of inversions is $O(t^2 k^2)$. CRT requires $\text{mod } M$ reduction which has a complexity of $O(\log M)^2 = O(t^2 k^2)$. The original algorithm of Garner [5] required only $t-1$ inversions but required a considerably larger number of modular reductions $O(k^2)$ and residue vector operations of the $O(t)$. The overall complexity of GA can be shown to be $O(t^2 k^2)$. The ART algorithm developed in next section requires only $t-1$ inversions. Also as in GA, it does not require $\text{mod } M$ reduction and thus has a complexity is $O(tk^2)$ bit operations.

5 ART-Algorithm

The underlying principle behind the Aryabhata's solution for the problem of two residues is that it requires only one modular inverse operation and any modular reduction is to the smaller moduli m_i rather than to composite M . This simplicity is of paramount importance. This principle has been exploited in many applications and the performance, for instance, of RSA signature has improved for smart-card processors by a factor greater than 3.6 [9, 10]. ART-algorithm is an extension of this principle to t moduli.

$$X = \text{ART}(v_1, v_2, \dots, v_t; m_1, m_2, \dots, m_t; M)$$

$$\text{Step 1 : } X_1 = v_1$$

$$\text{Step 2 : } X_2 = \text{ART}(v_1, v_2; m_1, m_2; M_2) \quad M_2 = m_1 m_2 \\ = \text{ART}(0, |v_2 - v_1|_{m_2}; m_1, m_2; M_2) + v_1$$

$$\text{Step 3 : } X_3 = \text{ART}(X_2, v_3; M_2, m_3; M_3) \quad M_3 = m_1 m_2 m_3 \\ = \text{ART}(0, |v_3 - X_2|_{m_3}; M_2, m_3; M_3) + v_2$$

.....

$$\text{Step } t : X_t = \text{ART}(X_{t-1}, v_t; M_{t-1}, m_t; M_t) \quad M_t = M \\ = \text{ART}(0, |v_t - X_{t-1}|_{m_t}; M_{t-1}, m_t; M_t) + v_t$$

The algorithmic form of the above is as follows:

INPUT : a positive integer $M = \prod_{i=1}^t m_i$ with $gcd(m_i, m_j) = 1$ for all $i \neq j$, and a modular representation $v(x) = (v_1, v_2, \dots, v_t)$ of x for the m_i .

1. $N_1 \leftarrow 1, X_1 \leftarrow v_1$
2. For i from 2 to t do the following:
 - $N_i \leftarrow N_{i-1} \cdot m_{i-1}$
 - $C_i \leftarrow N_i^{-1} \bmod m_i$ (also denote $|N_i^{-1} / m_i|$)
 - $u_i \leftarrow [(v_i - x_{i-1}) \cdot C_i] \bmod m_i$
 - $X_i \leftarrow X_{i-1} + u_i \cdot N_i$

Output : Return X_t

This is illustrated by example below:

Find $X = ART(2, 1, 3, 8; 5, 7, 11, 13; 5005)$

i	N_i	$N_i \bmod m_i$	C_i	U_i	X_i
1	1	--	--	--	2
2	5	5	$ 5^{-1} _7 = 3$	$ 1-2 _7 = 3$	$2+4 \cdot 5 = 22$
3	$5 \cdot 7 = 35$	$ 35 _{11} = 2$	$ 2^{-1} _{11} = 6$	$ 3-22 _{11} = 7$	$22+7 \cdot 35 = 267$
4	$35 \cdot 11 = 385$	$ 385 _{13} = 8$	$ 8^{-1} _{13} = 5$	$ 8-267 _{13} = 5$	$267+5 \cdot 385 = 2192$

The Step 2, 3, and 4 in the Table are iteration of the ART, solving for 2 residues in each of these steps. The final value $X = X_4 = 2192$.

6 Conclusion

The underlying principle behind the Aryabhata's solution for the problem of two residues and its simplicity are of paramount importance. Historians of mathematics have acknowledged this fact by writing about Aryabhata Algorithm (Kak 1986, [1]), but as part of the cryptology community, we are now trying to redress this balance. This principle has been reinvented and quite independently by Garner and exploited in many applications by others [9,10]. The performance, for instance, of *RSA* signature using this principle has improved for smart-card processors by a factor greater than 3.6. However Aryabhata has not been recognized for this contribution where ever *CRT* is mentioned. We bring about this fact and thus we give long over due credit to a great mathematician. We provided here Aryabhata

Remainder Theorem as an extension to t moduli of his original contribution. Its complexity is shown to be comparable or better than CRT and GA.

References

1. S. Kak, "Computational Aspects of the Aryabhata Algorithm," *Indian Journal of History of Science*, 21(1), pp. 62-71, 1986.
2. W.E. Clark, *The Aryabhata of Aryabhata*, University of Chicago Press, Chicago, 1930.
3. K.S. Shulka and K.V. Sarma, *Aryabhateeya of Aryabhata*, Indian National Science Academy, 1980.
4. C.F. Gauss, *Disquisitiones Arithmeticae*, 1801. English translation by Arthur A. Clarke, Springer-Verlag, New York, 1986.
5. H. Garner, "The residue number system", *IRE Transactions on Electronic Computers*, EC-8, pp. 140-147, 1959.
6. D.E. Knuth, *The Art of Computer Programming – Seminumerical Algorithms*, volume 2, Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981.
7. A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography* (CRC Press Series on Discrete Mathematics and Its Applications), 1996.
8. I.G. Pearce, *Indian Mathematics: Redressing the Balance*, <http://www-history.mcs.st-andrews.ac.uk/history/Projects/Pearce/index.html>
9. H. Handschuh and P. Paillier, "Smart Card Crypto-Coprocessors for Public-Key Cryptography," *CryptoBytes*, Vol. 4, No. 1, pp. 6-11, 1998.
10. *Public-Key Cryptography Standards, PKCS#1*, Version 2.1, RSA Laboratories, <http://www.rsasecurity.com/rsalabs/pkcs>
11. C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World*, 2nd edition, Prentice Hall, 2002.
12. Srinivas Iyengar, C.N., "The History of Ancient Indian Mathematics", The World Press, Chicago, 1930.
13. T.R.N. Rao and C.H. Yang, "Aryabhata Remainder Theorem: Relevance to Public-Key Crypto-algorithms," Technical Report TR-2004-0-1, Center for Advanced Computer Studies, University of Louisiana at Lafayette, August 2004. http://crypto.nknu.edu.tw/publications/art_report.pdf