*Intern. J. Computer Math.*, Vol. 63, pp. 215–226
Reprints available directly from the publisher
Photocopying permitted by license only



# A FAST MODULAR EXPONENTIATION FOR RSA ON SYSTOLIC ARRAYS

YONGFEI HAN*, C. J. MITCHELL and D. GOLLMANN

*Dept. of Computer Science, Royal Holloway, University of London*
*Egham, Surrey TW20 OEX, U. K*

*(Received 4 April 1995)*

This paper presents two systolic algorithms for modular exponentiations based on a $k$-SR representation. In a systolic $k$-SR scheme, throughput is one modular exponentiation of a message block having $n$ digits in every clock cycle, with a latency of nearly $5n/4$ cycles to output the final result. The speedup for a group of messages having $l$ message blocks is around $\left(\frac{5}{6l}+\frac{2}{3n}\right)$, compared to a single processor or processing element for modular multiplications. The scheme saves nearly $\frac{n}{4}$ processing elements and around $\frac{n}{4}$ modular multiplications, compared with the scheme in [23].

*Keywords:* Cryptography; RSA; modular exponentiation; systolic arrays

*C.R. Categories*: F1.2, 1,2, E.3

## 1. INTRODUCTION

Modular exponentiations of integers are basic operations for several well known cryptographic algorithms in public-key cryptosystems such as the RSA scheme [18]. The security of these cryptosystems is based on the difficulty of factoring integers. To achieve enough security, the word lengths and key lengths in the modular exponentiations should be significantly greater than those used in conventional general-purpose computer hardware. The currently required typical word length is around 512 bits or more, and it will grow in future as the cryptanalysis makes progress.

*The first author is supported by the ORS Award and the DTI/EPSRC LINK Personal Communications Programme project under UK EPSRC research grant GR/J17173.

The requirement on the lengths makes RSA slow. The encryption and decryption rate is much slower than with symmetric algorithms. The expert group in the DECT system ruled out public key cryptographic algorithms in the implemented telecommunication system since they are too complex in computation, although they are an attractive option. While the DECT system requires the cryptographic algorithm to execute in 100–200 ms, the RSA implementation took around 1–2 seconds on an Intel 80286 level microprocessor. GSM also ruled out public key cryptographic algorithms [13]. When bulk data are transmitted in mobile telecommunication systems, cryptographic algorithms are also required to be fast and cheap for the encryption and decryption of bulk data. Hence it is quite natural to speed up modular exponentiation.

Many efforts have been made to speed up public key cryptographic algorithms. One of them is to reduce the computational complexity of the algorithms. Much meaningful research work has been reported [12,14,17]. Another is to utilize parallel techniques to perform faster implementation [3,4,21]. Systolic arrays [16], as a parallel means, have been applied in many fields [7,8,5,9,10]. The central point in the systolic approach is to ensure that once an information item is brought into the system it can be used effectively and repetitively while it is being 'pumped' from cell to cell through the system. Combinations of multiprocessing and pipelining are the crux of the systolic approach of parallel processing.

Some systolic algorithms for public key cryptography have appeared. Koç and Hung [2] presented a systolic algorithm for modular multiplications which suffers from excessive latency and a slow clock. Shand, Bertin and Vuillemin [20] described a pipeline which is similar to one row of the systolic array presented by Walter [22] where the scheme is only for modular multiplications and the latency of $2n + 2$ clock cycles is much higher. Note the clock cycles in systolic arrays for modular multiplications are different with clock cycles in following modular exponentiations.

A linear systolic array implementation for RSA based on a binary algorithm was proposed by Zhang and Yun [24]. Further, Zhang presented a systolic array implementations for RSA based on a signed-digit representation [23]. The first of them needs $2n$ processing elements. The second requires $\frac{3n}{2}$ processing elements. In their schemes, however, $M^{-1}$ is brought into every cell but is not used effectively, that is, $M^{-1}$ is not used for computations in many cells although it moves through each cell.

The systolic system presented here proposes a systolic modular exponentiation based on $k$-SR representations and fast modular multiplications. Our scheme is based on the fact that the $k$-SR algorithm is faster

than the signed-digit algorithm because it needs less modular multiplications and no precomputation of $M^{-1}$.

## 2. BASIC ALGORITHMS

In the RSA cryptographic algorithm, first randomly find two large primes $p$ and $q$ and define $N = pq$. $M$ is a message to be transmitted as an integer in the range $1, \ldots, N-1$. $M$ has to be broken into blocks if it is too big. Such a block is called a message block in this paper. Encrypt $M$ into a cryptogram $C$ by the rule

$$C = M^e (\text{mod } N)$$

and decrypt by using the *private key* $d$ and the formula

$$D = C^d (\text{mod } N)$$

Both encryption and decryption are modular exponentiations. The common method for performing modular exponentiations is the 'square and multiply' algorithm [15]. This algorithm is briefly described as follows. If the computation

$$M^e (\text{mod } N)$$

is required and $e$ has the binary representation

$$e_{n-1} e_{n-2} \cdots e_0$$

where $e_{n-1}$ is the most significant bit, then the 'left to right' version of the algorithm works as follows:

*Algorithm: Square and Multiply*

$x = 1$;
for $(i = n-1;\ i \geq 0;\ i\ --)\{$
    $x = x^2 (\text{mod } N)$;
    *if* $(e_i = 1)$
    $x = x \times M\ (\text{mod } N)$;

Mitchell [6] proposed Canonical $k$-SR representations for integers. In this representation, we use coefficients $2^i - 1, 0 \leq i \leq k$. The representation converts $i$ consecutive ones to the coefficient $2^i - 1$ for any $i$ satisfying $2 \leq i \leq k$, where $k$ is fixed. The converting procedure starts from the most significant digit replacing every string of $i$ consecutive ones with a string of

$i-1$ zeros in the $(i-1)$ most significant positions and $2^i-1$ in the least significant position. The definition of the least significant bits and the most significant bits in the paper follows ISO/IEC 9797 [1].

Now let $e$ have an $k$-SR representation

$$f(r-1)f(n-2)\ldots f(0)$$

where $f(r-1)$ is the most significant digit and $f(i) \in (0,1,3,7,\ldots,2^k-1), i<r$. Combining 'square and multiply' with precomputations of $m^3,\ldots,m^{2^k-1} \pmod N$, then, a modified 'left to right' version of the square and multiply algorithm with $k$-SR representations works as follows:

*Algorithm: Square and Multiply with k-SR representations*

$x = m^{f(r-1)} \pmod N$;
for $(i = r-2;\ i \geq 0;\ i--)\{$
  $x = x^2 \pmod N$;
  if$(f(i) \neq 0)$
  $x = x \times M^{f(i)} \pmod N$;

The algorithm reduces the number of modular multiplications to about $n/4$ when $k$ becomes large [6].

## 3. SYSTOLIC ARRAY AND ALGORITHMS

We first design a cell (processing element) for the systolic arrays.

DEFINITION 1 *A systolic cell is defined as a function in which:* $D_{\text{out}} = \phi(D_{\text{in}}, B, f_s(i))$. $B \in (M, M^3, M^7, \ldots, M^{2^k-1} \bmod N)$. $f_s(i)$ *is a control function.*

DEFINITION 2 *In a cell,*

$D_{\text{out}} = D_{\text{in}}^2 \bmod N$ *if* $f_s(i) = 0$.
$D_{\text{out}} = D_{\text{in}} \times M^{(2^{f(i)}-1)} \bmod N$ $f_s(i) \geq$
$D_{\text{out}} = 0$ *if* $f_s(i) = \alpha$.

$\alpha$ denotes an empty operation as the input symbol.

The framework of the cell is shown in Figure 1 where 're' is a register. The precomputed value or $M$ is fed into the cell through $B$. $D_{\text{in}}$ is fed by the output result of the left-cell through $D_{\text{out}}$. Hence, in a cell, only one modular

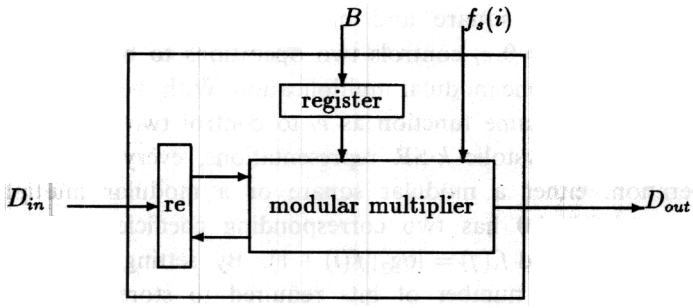


FIGURE 1 The framework of a cell in systolic.

multiplication or modular square can be computed since modular squares can be computed in a modular multiplier.

We then design a linear systolic array in Figure 2 to execute the encryption and decryption of RSA. $B \in (M, M^3, M^7, \ldots, M^{2^k-1} (\text{mod } N))$ is fed into the cells from the top. The signal $f_s(i)$ is a control function of the operations inside the cells. For a fixed exponent, $f_s(i)$ is also fixed and there is really one value $B$ can take. No operation will be done if $f(i)$ is $\alpha$ as defined in Definition 2.

Suppose the $k$-SR representation is $f(r-1)f(r-2)\ldots f(0)$. We design an algorithm to derive a systolic $k$-SR representations $f_s(r+q-2)f_s(r+q-3)\ldots f_s(0)$, where $q$ is respectively defined as

DEFINITION 3 *$q$ is the number of non-zeros in a $k$-SR representation* $f(r-1)\ldots f(0)$.

We make use of $q$ and $r$ to analyse the computational complexity.

The idea in the systolic algorithm is to perform a modular square or modular multiplication with a $f_s(i)$, which is different from the binary representation and $k$-SR representation. The algorithm converting to systolic $k$-SR representation scans $f(r-1)f(r-2)\ldots f(0)$ starting from $f(0)$. If $f(i)=0, f_s(j)=0$. If $f(i)\neq 0, f_s(j+1)=0$, and $f_s(j)=\log_2(f(i)+1)$.

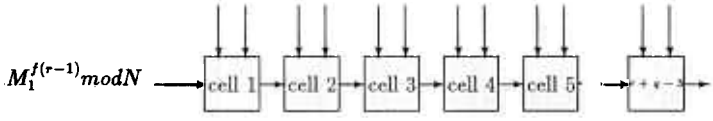


FIGURE 2 A linear systolic array.

Note that in the square and multiply algorithm with a binary representation, if $e_i \neq 0, e_i$ controls two operations to be performed: one modular square and one modular multiplication. With the $k$-SR representations, $f(i)$ has the same function as $e_i$ to control two operations to be performed. In the systolic $k$-SR representations, every $f_s(j)$ determines one operation: either a modular square or a modular multiplication. Hence, every $f(i) \neq 0$ has two corresponding coefficients $f_s(j+1)f_s(j)$ with $f_s(j+1) = 0$ and $f_s(j) = \log_2(f(i)+1)$. By setting $f_s(j) = \log_2(f(i)+1)$, we reduce the number of bits required to store $f(i)$ from $k$ to $\log_2(k+1)$. The most significant position $f(r-1)$ does not have to be split into two operations and corresponds to a single coefficients $f_s(r+q-2)$.

*Algorithm: Converting to Systolic k-SR Representations*

$i = 0;$
$j = 0;$
while $(i < r-2)$ do {
    if $f(i) = 0$ then $\{f_s(j) = f(i); j = j+1;\}$
    if $f(i) \geq 1$ then $\{f_s(j) = \log_2(f(i)+1); j = j+1; f_s(j) = 0; j = j+1$
$f_s(j) = f(r-1);$

Suppose $l$ is the number of message blocks. A $k$-SR systolic algorithm first computes $M_i^{f_s(r+q-2)}, 1 \leq i \leq l, f_s$ denotes a systolic $k$-SR representation. Then it performs the square and multiply.

*Systolic k-SR Algorithm*

for $(i = 1; i \leq l; i++)$
    $\{x_i = M_i^{f_s(r+q-2)} \bmod N;$
$i = 1;$
$j = r+q-3;$
pipeline for $x_i$ {
while $j \geq 0$ and $i \leq l$ do {
    if $f_s(j) = 0$
        then $x_i = x_i^2 \bmod N$
    if $f_s(j) \geq 1$
        then $x_i = x_i \times M^{f(i)} \bmod N$

$j = j - 1;$

end the pipeline

In RSA encryption and decryption, $e$, $d$ and $N$ are fixed. Suppose that $l$ message blocks $M_i$, $i = 1, \ldots, l$ have to be encrypted or decrypted. The exponents $e$ and $d$ can be converted to their $k$-SR representation. Then the $k$-SR representation is converted to the systolic $k$-SR representation. Values of precomputations are stored in a group of local memories denoted by $LM$ where each local memory is connected with a cell. That is, $r + q - 3$ local memories are working with a common clock as $r + q - 3$ cells are working with the common clock. A message block is fed to $D_{\text{in}}$ of cell 1 in one clock cycle. In every common clock cycle, $D_{\text{out}}$ of cell $i$ is forwarded to $D_{\text{in}}$ of cell $i + 1$; output of $LM$ $i$ is forwarded to $LM$ $i + 1$. Every common clock cycle may be divided into several clock cycles for $LM$. Several values of precomputations could be transmitted between $LM$ $i$ and $LM$ $i + 1$ while a modular multiplication is done in cells in one common clock cycle. Since we are working on the level of algorithms, the paper does not discuss further the detail of the bandwidth and how to produce a chip.

Figure 3 shows the linear systolic array runing in the third clock cycle, where $p_2$ is $f_s(r+q-2)f_s(r+q-3)f_s(r+q-4), p_1$ is $f_s(r+q-2)f_s(r+q-3).p_0$ is $f_s(r + q - 2)$.

The whole systolic system works as a pipeline. The precomputed exponents $M^{2^i-1} (i \leq k) \bmod N$ are stored in a group of local memories which can send data to cells from the top. A host system sends $M_i (i \leq l)$ to the first cell from the left. $f_s(r + q - 3)$ controls operations inside cell 1. If $f_s(r + q - 3) \neq 0$, a precomputed value depending on the value of $f_s(r + q - 3)$

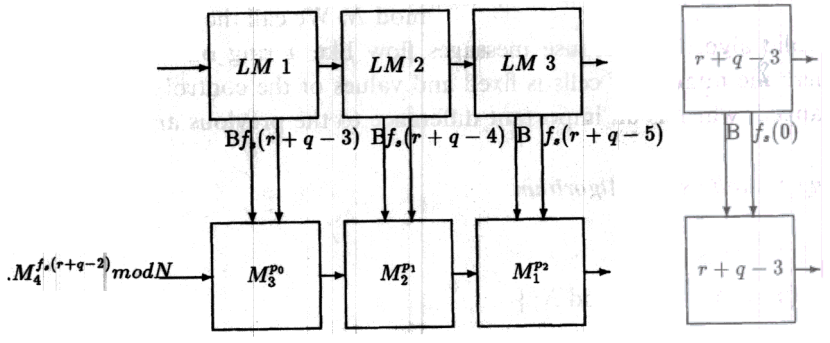


FIGURE 3 In third clock cycle.

is fed into cell 1 through $B$ to perform a modular multiplication operation. It is not necessary that the host system must be a computer. In the case of real-time signal processing, e.g. cable TV signal, systolic systems are suitable for devices accepting encrypted signals and then passing them on to a systolic array. The precomputation of $M^{2^i-1} \bmod N$ can be performed by a host system or a computer attached to the local memories. Then these values are distributed to every cell by local memories.

In the $k$-SR representation, the $k$ is fixed in a practical system. The input $f_s(i)$ to a cell is set to $\alpha$ before $M_1$ or its exponentiations reaches the cell. The whole array is synchronized by a common clock to compute the sequence $M_i^e \pmod N$ or $M_i^d \pmod N$. It will take $r+q-3$ clocks for the first result to appear in the output of the array, after the input data flows into the array. The last message block $M_l$ will come out of the array in $(r+q+l-3)$-th clocks. This time period is referred to as computation time.

In bulk data transmissions, $l$ is quite large. If $l \gg q+r-3$, the systolic system has high load because the systolic system is full after the $(r+q-2)$- th clock cycle and before the $l-(r+q-2)$-th clock cycle. In some cases of digital transmissions, $l$ is not large and the system has lower load. We work out a flexible algorithm suitable for these cases.

DEFINITION 5

$$m = \left\lceil \frac{r+q-3}{n_1} \right\rceil, \quad 0 < n_1 \leq \frac{r+q-3}{2}$$

We divide $f_s(r+q-3) \ldots f_s(0)$ into $n_1$ groups where the length of each group is $m$. Hence there are $m$ cells in a systolic array. The second group $f_s(r+q-3-m) \ldots f_s(r+q-4-2m)$ controls the computation of messages after the first group $f_s(r+q-3) \ldots f_s(r+q-4-m-1)$ computed $M_l^{f(r-2)\ldots f(r-3-m)}, \ldots, M_1^{f(r-2)\ldots f(r-3-m)} \bmod N$. We call the algorithm the ring systolic algorithm because messages flow like a ring on a systolic array where the number of cells is fixed and values of the control signal $f_s(i)$ are changed, which is an important difference to the previous array.

*Ring Systolic k-SR Algorithm*

```
for (i = 1; i ≤ l; i++)
    {x_i = M_i^{f_s(r+q-2)} mod N;}
i = 1;
j = r + q + 3;
```

pipeline for $x_i$\{
while $j \geq r_1 + q - m - 4$ and $i \leq l$ do
if $f_s(j) = 0$
then $x_i = x_i^2 \bmod N$:
if $f_s(j) \geq 1$
then $x_i = x_i \times M^{f(i)} \bmod N$;
$j = j - 1$;
}
} end the pipeline

$i = 1$;
$j = r + q - 3 - (n_1 - 1)m$;
pipeline for $x_i$ {
while $j \geq 0$ and $i \leq l$ do {
if $f_s(j) = 0$
then $x_i = x_i^2 \bmod N$:
if $f_s(j) \geq 1$
then $x_i = x_i \times M^{f(i)} \bmod N$;
$j = j - 1$;

} end the pipeline

It will take $(l - m) * (n_1 - 1) + m * n_1$ clocks for the final result of $M_1$. The final result of the last message block $M_l$ will come out of the array in the $(l - m) * (n_1) + m * n_1$-th clock cycle. For a message block, the computation time is $(l - m) * (n_1 - 1) + m * n_1$.

## 4. COMPARISON WITH OTHER TECHNIQUES

Based on $k$-SR representations, the systolic $k$-SR algorithm for the linear systolic array implementation of RSA requires about $\frac{5n}{4}$ cells. Note that $r$ is very close to $n$, and $q$ is close to $\frac{n}{4}$ in an ideal situation, so we may ignore the difference between $r$ and $n$, and $q$ and $\frac{n}{4}$. Hence, the latency of a modular exponentiation of a message block is $\frac{5n}{4}$. The product of time and area is $O\left(\frac{25n^2}{16}\right)$. Compared to Zhang's scheme, the systolic $k$-SR scheme saves about $\frac{n}{4}$ cells. The speedup is also around $\frac{n}{4}$ time. The ring systolic $k$-SR scheme could save more than $\frac{n}{4}$ with increased time consumed.

Suppose the latency of the modular multiplication is $T$. The computation time for $l$ message blocks is $\frac{3}{2}nlT$ on average in the square and multiply

TABLE I Comparison of Algorithms

| *Algorithm* | *Cells* | *Load rate* | *Precomp* | *Number of bits* | *Latency* |
|---|---|---|---|---|---|
| | | $\frac{l}{l+\frac{5n}{4}}$ | $k-1S+k-1M$ | $2n+\log_2 k+1$ | |
| | | $\frac{l}{l+3n}$ | $M^{-1}$ | $2n+3$ | |
| | | $\frac{l}{l+2(m-1)}$ | $k-1S+k-1M$ | $2n+\log_2 k+1$ | $(l-m)*(n_1-1)+n_1 m$ |
| | | $\frac{l}{l+4n}$ | | $2n+2$ | $2n$ |

algorithm with binary representations using a single processor or cell. In systolic $k$-SR scheme, it is $\frac{5}{4}nT+lT$. So we have

$$\frac{\frac{5}{4}nT+lT}{\frac{3}{2}nlT}=\frac{5}{6l}+\frac{2}{3n}$$

From [19], the fastest VLSI hardware implementation for RSA is about 1000 times slower than DES. In software, DES is about 100 times faster than RSA. The systolic $k$-SR scheme can be about 1000 times faster than the fastest VLSI hardware implementation for RSA when $l$ is 100000 and $n$ is 672 with 840 cells.

The ring systolic $k$-SR scheme could reach the speed with less cells and more message blocks. The computation time for $l$ message blocks is $l*n_1$ in the scheme. Hence we have

$$\frac{ln_1T}{\frac{3}{2}nlT}=\frac{2n_1}{3n}$$

When $n$ is 600, and $n_1$ is 3, the scheme could speed up 300 time with 260 cells, compared with a single faster RSA processor.

A comparison of these schemes with the schemes presented by Zhang [23], and Zhang and Yun [24] is listed in Table I where $k-1\,S$ and $k-1\,M$ is $k-1$ squares and $k-1$ multiplications.

## 5. CONCLUSION

Two systolic $k$-SR algorithms and a linear systolic array for modular exponentiations have been presented. The array has simple, regular communication and control structures, and is thus very suitable for very-large-scale integration implementation. The proposed algorithms employ the $k$-SR representation which significantly reduces the number of modular

multiplications in RSA, and also reduces the number of cells in systolic arrays. The schemes is suited to RSA cryptography in image transmissions, also suitable for other bulk data. Further, it achieves better systolic design measurement in terms of the product of area and time. Unlike other schemes proposed in [2,20,22,23,24], the ring systolic $k$-SR scheme could be implemented using currently available technology while a common clock cycle consists of $2^i (i \geq 3)$ clock cycles so that each clock cycle drives about 100 to 200 bits to a cell. The systolic algorithms is also suited to signed-digit representations, minimal $k$-SR representations [11] and $SS(l)$ representations.

*Acknowledgments*

The first author Yongfei Han would like to thank Prof. D.J. Evans for his help on systolic algorithms when he was with Loughborough University of Technology.

*References*

[1] ISO/IEC 9797. Information technology – security techniques – data integrity mechanism using a cryptographic check function employing a block cipher algorithm. Second edition, 1994-04-15.

[2] Koç, C. K. and Hung, C. Y. (1991). Bit-level systolic arrays for modular multiplication. *J. VLSI Signal Processing*, **3**, pp. 215–223.

[3] Eldridge, S. E. and Walter, C. D. (1993). Hardware implementation of montgomery's modular multiplication algorithm. *IEEE Trans. Computers*, **42**(6), pp. 693–699.

[4] Eldridge, S. E. (1991). A faster modular multiplication algorithm. *International Journal of Computer Mathematics*, **40**, pp. 63–68.

[5] Evans, D. J. (1991). *Systolic Algorithms*. Gordon and Breach Science Publishers.

[6] Gollmann, D., Han Yongfei and Mitchell, C. (1996). Redundant integer representations and fast exponentiation. *Designs, Codes and Cryptography*, **7**, pp. 135–151.

[7] Han Yongfei and Evans, D. J. (1994). Parallel inference algorithms for the connection method on systolic arrays. *International Journal of Computer Mathematics*, **53**(34), pp. 177–188.

[8] Han Yongfei (1983). Simulation systems for Bit-Slice VLSI. *Journal of Computer Engineering* (*Chinese*) No. **4**.

[9] Han Yongfei and Evans, D. J. (1995). On systolic arrays and neural nets for inference. *International Journal of Computer Mathematics*, to appear.

[10] Han Yongfei and Evans, D. J. (1992). The simulation methodology using models and parallel languages. *Proc. of the 92 Summer Computer Simulation Conference*. pp. 58–61. Reno, Nevada, USA.

[11] Han Yongfei Mitchell, C. and Gollmann, D. *Minimal k-SR Representations*. Technical Report in preparation, Dept. of Computer Science, Royal Holloway, Uni. of London.

[12] Hui, L. C. K. and Lam, K. Y. (1994). Fast square-and-multiply exponentiation for RSA. *Electron. Lett.*, **30**(17), pp. 1396–1397.

[13] European Telecommunications Standards Institute. ETS 300 175–7 Radio Equipment and Systems (RES); Digital European Cordless Telecommunications (DECT) Common interface Part 7: Security feature. (1992).

[14] Jedwab, J. and Mitchell, C. J. (1989). Minimum weight modified signed-digit representations and fast exponentiation. *Electronics Letters*, **25**(17), pp. 1171–1172.
[15] Knuth, D. E. (1981). *The art of computer programming*, Volume **2**: seminumerical algorithms. Addison-Wesley, Reading, Mass., 2nd edition.
[16] Kung, H. T. and Leiserson, C. E. (1978). Systolic array for (vlsi). *Proc. of the Symposium on Sparse Matrices Computations*, I.S. Duff et al. (eds.), *Knoxville, Tenn.*, pp. 256–282.
[17] Lam, K. Y. and Hui, L. C. K. (1994). Efficiency of SS(1) square-and-multiply exponentiation algorithms. *Electron. Lett.*, **30**(25), pp. 2115–2116.
[18] Rivest, R. L., Shamir, A. and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communication of the ACM*, **21**, 120–126.
[19] Bruce Schneier, Applied cryptography - protocol, algorithms, and source code in c.
[20] Shand, M., Bertin, P. and Vuillemin, J. (1991). Hardware speedups in long integer multiplication. *ACM Sigarch*, **19**, pp. 106–113.
[21] Walter, C. D. (1994). Logarithmic speed modular multiplication. *Electronics Letters*, **30**(17), pp. 1397–1398.
[22] Walter, C. D. (1993). Systolic modular multiplication. *IEEE Trans. Computers*, **42**(3), pp. 376–379.
[23] Zhang, C. N. (1993). An improved binary algorithm for RSA. *Computer Math. Applic.*, **25**(6), pp. 15–24.
[24] Zhang, C. N. and Yun, D. D. Y. (1988). A parallel algorithm and systolic design for RSA cryptosystem. *International Conf. on Systolic Arrays*, San Francisco, CA, pp. 233–237.