An efficient implementation of rolling array based Stream cipher in FPGA

Panneerselvam Ramadoss§, Suganya Annathurai‡, NallaAnandakumar N‡, Muthaiah R§
§ School of computing, Sastra University, Thanjavur
‡ Hardware Security Research Group, Society of Electronics Transactions and Security(SETS), Chennai, India
Email{panneerselvam.sastra}@gmail.com

Abstract—In this paper, we investigate the efficient implementation of rolling array based stream cipher in FPGA. Rolling array is the rotation of each element in an array for each iteration. Rotation means shifting of values to consecutive index. Array rotation can be efficiently implemented in software, whereas, it is time and area consuming process in hardware like FPGA and ASIC. This paper analyze various methodologies to implement rolling array and propose the efficient way, in terms of area and time by comparing the performance in FPGA. To verify the efficiency of methodologies, we applied all methodologies in a cryptographic algorithm RCR32 (Rolling, Constant Rotation- 32 bit) rolling array based stream cipher and presented its performance in this paper. We use VHDL (Very High Speed Integrated Circuit Hardware Descriptive Language) for implementation and the target device is Xilinx Virtex-4 FX-60 FPGA.

I. INTRODUCTION

ROLLING array is similar to rotor machine, basic principle of rolling array is when array is rotated the values are shifted to consecutive position and first word will be shifted to last position, if it is rotated for many times the values will be rotating all over the array. Advantage over static array, rolling array produces different data while accessing the same address in consecutive iteration because for each iteration the array is rotated, whereas static array produces the same data for all iteration. This usefull property of rolling array is used in the cryptographic algorithm like enigma machine, RC4 and Py [1].

Stream ciphers encrypt one bit plain text at a time by using encryption function. It is a symmetric key cipher where both encryption and decryption process uses the same key. The process of encryption/decryption is plaintext/cipher text is exclusive-ored with a pseudo random key stream, generated using shared symmetric key. Pseudo random key stream generator can be contructed using LFSR(Linear Feedback Shift Register) based, rolling array based. Stream cipher like Grain, E0, A5/1 and W7 uses LFSR. Stream ciphers like RC4, Py, RCR32 are rolling array based stream ciphers. These rolling arrays are rotated and updated for every keystream.

RCR32 is derived from Py cipher [1] is the candidate for e-Stream contest, round function of Py algorithm is slightly modified and RCR32 is proven to be more secure than Py [2]. RCR-32 algorithm is designed to be faster in software than in hardware, however, we had taken this algorithm and explored the efficient way of implementation, yet array rotation seems to be challenging in the hardware. We discussed the comparative report for efficient ways of implementing array rotation. Various ways of array rotation had been applied in RCR32 algorithm on the whole and also we have applied the same only in round function to analyze the impact in throughput of the algorithm.

II. OUR CONTRIBUTION

Array of elements usually occupies more logic space in FPGA. If the array is operated frequently for updation and rotation, then the performance of the application will be affected. If the array size is large for example 260 elements and each element of size 32-bits then the area and time for array operation increases. Focus of this work is to analyse different methodologies of implementing rolling array in FPGA. All methodologies are applied to a stream cipher and its performance is analaysed and presented. To the best of our knowledge, we have not encountered any previously published implementations of rolling array in FPGA.

III. IMPLEMENTATION OF ROLLING ARRAY

Three methods of array rotation had been attempted, which are as follows:

A. Method 1

Shift each element in the array for every clock cycle. The first element of the array is shifted to the last element of the array, second element is shifted to first, third element shifted to second position and so on. Hence for array size of n requires n+1 clock for one rotation. Since this method uses single array for its operation, this method requires lesser area and more time. For example, an array array of size 5 elements, takes 6 clock cycles to perform one rotation.

B. Method 2

In this method, the array is copied to the temporary array in first clock cycle and the index of the temporary array is changed and copied to original array in the next clock cycle as shown in Fig 1. Therefore it takes two clock cycles to perform one array rotation. Since this method uses two arrays, it requires more area and lesser time. In VHDL, without using the temporary array, the same array is shifted with new index
in single clock cycle as shown in Fig 2 and it was tested in the board level.

Example1: - An array Y of size 5 elements, is copied to the temporary array in one clock and for the next clock the index is changed and shifted to the Y array as shown in Fig 1, so it requires two clock cycles for a single rotation.

![Fig. 1. Temporary array based rotation](image)

Example 2: - An array Y of size 5 elements, in VHDL it is possible to copy the same array in one clock by changing the index can perform array rotation as shown in Fig 2. As an extra temporary array is not used, the circuit can be implemented with reduced area and high speed.

![Fig. 2. Index based rotation](image)

**C. Method 3**

This method is based on index access rather than physically rotating the array. For example an array Y(0) to Y(5) after one rotation value of Y(1) will be shifted in Y(0). Therefore incrementing the pointer value by one Y(ptr) can access the value in Y(1). This operation requires only the access time, which is only one clock cycle.

For example, consider the array Y of size 5, for each iteration index is incremented after accessing the $i^{th}$ element index become zero by the mod operation as shown in Fig 3.

![Fig. 3. Pointer based rotation](image)

An array size of 260 elements each of 32 bits word is chosen for analysis, as RCR32 uses the same size array. All three methods are implemented with random values as inputs to the array. Table 1 shows the comparative study for three methods of array implementation in Virtex-4 FPGA. It is observed that method 1 occupies moderate area but more time for one rotation operation, method 2 takes one clock but occupies comparatively large area and method 3 is very efficient in terms of clock and area.

![Fig. 4. Performance Chart](image)

**Table I**

<table>
<thead>
<tr>
<th></th>
<th>Method 1</th>
<th>Method 2</th>
<th>Method 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock cycles</td>
<td>260</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Slice %</td>
<td>28</td>
<td>20</td>
<td>0.18</td>
</tr>
<tr>
<td>Number of Slices</td>
<td>7100</td>
<td>5070</td>
<td>48</td>
</tr>
</tbody>
</table>

**IV. APPLYING ARRAY ROTATION IN RCR-32**

To verify the observation made in Section III, cryptographic algorithm RCR32 stream cipher which is based on the rolling array is taken. RCR32 is using rolling array as major component, in this algorithm array is not only rotated, it is also updated, hence influence of array rotation in overall performance of the algorithm is high.

RCR-32 has three modules key-schedule, IV-schedule and round function. Key-schedule and IV-schedule are same as Py cipher [1]. Key-schedule receives 128 bits symmetric key. It uses Internal Permutation (IP) of $256 \times 8$ and vector S of 32 bits. This module initializes Y array of size $260 \times 32$.

IV-schedule has two parts namely IV-part-I and IV-part-II. IV-part-I receives Y array from key schedule and IV (Initialization Vector) of 128 bits. This module initializes a new permutation array P of size $256 \times 8$ and EIV of size $16 \times 8$. This module determines the new vector S which depends on P. In the IV-part-I, values of the array Y is not altered. These three arrays Y array of size $260 \times 32$ (260 elements of each 32 bits), P array of size $256 \times 8$ and EIV array of size $16 \times 8$ are rotated and updated for 260 times in IV-Part-II.

In round function, for every keystream word (32 bits) Y and P array are updated and rotated. In round function Y array will be indirectly accessed by P array, like Y (P(72)).
A. Issues of RCR32 while applying different array rotation

In this section issues faced when applying three methods of array rotation in RCR32 algorithm are discussed.

Method 1 shifts one element for one clock cycle. One rotation of Y array (260 × 32) takes 261 clock cycles. In IV-part-II, Y array is rotated 260 times requires 67860 clock cycles. Though it is affordable delay, when considering that the IV-Part-II is one time process, this method affects the throughput of round function as well. The problem with huge number of clock cycles and less throughput is overcome in the method 2.

Method 2 uses redundant array based array rotation. Though it uses single array and requires one clock cycle for single rotation, this method occupies more logic space. More number of clock cycles and large area these problems are solved in Method 3.

Method 3 is a pointer based rotation. RCR32 algorithm uses an element in an array as the index to access the element in another array, for example Y(P(32)). This is solved by adding the element of inner array with the index pointer for example Y(P(72+indexp)+indexy). However, the size of two arrays are different so same pointer cannot be used for two arrays. After completing a cycle the index pointer has to be reset to zero, for that mod operation is used. In VHDL, modular function can be used for the values that satisfies $2^n$. Y array has 260 elements, therefore, mod operation cannot be used for resetting the pointer of Y array. This problem is overcome by subtracting the pointer value and element of inner array with 260 to get the correct index value to access Y array. This method takes less number of clock cycles and area. Table 2 shows the comparative study of RCR32 when applying different array rotation.

<table>
<thead>
<tr>
<th></th>
<th>Method 1</th>
<th>Method 2</th>
<th>Method 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock cycles</td>
<td>72513</td>
<td>3114</td>
<td>2861</td>
</tr>
<tr>
<td>Slice %</td>
<td>70%</td>
<td>74%</td>
<td>71%</td>
</tr>
<tr>
<td>Number of Slices</td>
<td>17,850</td>
<td>18,903</td>
<td>18,146</td>
</tr>
</tbody>
</table>

TABLE II
SYNTHESIS REPORT OF RCR32

V. IMPLEMENTATION OF ROUND FUNCTION

Throughput of RCR32 depends on the round function, so to verify the throughput of the round function Y and P array output from the key-schedule and IV-schedule are hard coded in the round function. All three methods are applied in the round function and its performance is analysed. It is implemented in FPGA Table 3 shows comparative report between slice, clock and throughput, therefore method 1 is worst case in terms of clock cycles, slice and throughput, method 2 having high throughput of 45 MB/Sec and method 3 is efficient occupying less slice, area and throughput 41 MB/Sec which is bit less than method 2.

<table>
<thead>
<tr>
<th></th>
<th>Method 1</th>
<th>Method 2</th>
<th>Method 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock cycles</td>
<td>528</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Slice %</td>
<td>50%</td>
<td>44%</td>
<td>29%</td>
</tr>
<tr>
<td>Number of Slices</td>
<td>12,847</td>
<td>11,248</td>
<td>7,381</td>
</tr>
</tbody>
</table>

TABLE III
SYNTHESIS REPORT OF ROUND FUNCTION

VI. CONCLUSION

In this paper, we analysed the efficient way of implementing rolling array and it is verified by applying it in RCR32 stream cipher algorithm. It is observed that, when the area and speed constrained environment Method 3 can be applied and in the throughput constrained environment Method 2 can be applied.

REFERENCES