RESEARCH ARTICLE OPEN ACCESS

High Security Masked AES Based On Retina Image as a Key

N. Balachandrarao, S. Parvathi Nair
Dept. of Electronics and Communication Engineering SRM University Chennai, India
Dept. of Electronics and Communication Engineering SRM University Chennai, India

Abstract
In order to protect "data-at-rest" in storage area networks from the risk of differential power analysis attacks without degrading performance, a high-throughput masked advanced encryption standard (AES) engine is proposed. However this engine adopts an unrolling technique which requires extremely large field programmable gate array (FPGA) resources. In this brief, we aim to optimize the area for masked AES with an unrolled architecture. We achieve this by mapping its operations from GF (2^8) to GF (2^4) as much as possible. We reduce the number of mapping for GF (2^8) to GF (2^4) and inverse mapping for GF (2^4) to GF (2^8) operations of the masked SubBytes step from ten to one. In order to be compatible, the masked Mixcolumns, masked AddRoundKey, and masked ShiftRows including the redundant masking values carried over GF (2^4). BRAM in FPGA block can be used to reduce hardware resources and with this we can achieve 40.9-Gbits/s.

KeyWords— Advanced Encryption Standard (AES), Masking, field programmable gate array (FPGA), Throughput

I. INTRODUCTION

All of the cryptographic algorithms we have looked at so far have some problem. The earlier ciphers can be broken with ease on modern computation systems. The DES algorithm was broken in 1998 using a system that cost about $250,000. It was also far too slow in software as it was developed for mid -1970's hardware and does not produce efficient software code. Triple DES on the other hand, has three times as many rounds as DES and is correspondingly slower. As well as this, the 64 bit block size of triple DES and DES is not very efficient and is questionable when it comes to security. with the help of NIST a brand new AES came into existence.

In 1999, Kocher *et al.* first broke the normal advanced encryption standard (AES) [1] by means of power analysis attacks. Later, the differential power analysis (DPA) attack was further developed as one of the most promising power analysis attacks. From then on, numerous efforts have been devoted to the development of efficient countermeasures for the AES implementations against DPA attacks. Two representatives are the multiplicative masking and the Boolean masking. They both try to remove the correlation between the power consumption and the secret keys. The multiplicative masking can be realized by using either standard CMOS cells at the gate level (which has been proved to be insecure in terms of glitch attacks) or nonstandard CMOS cells (which has been proved to be DPA resistant and glitch free but requires a semiautomatic design flow). On the other hand, the Boolean masking can be easily realized at the algorithmic level and is immune to DPA and glitch attacks. The Boolean masking has the advantage of easy implementation because it does not need extra specific hardware as in [3] and [4].

The Boolean masking is a good candidate to be applied to the AES in SANs, but if we directly apply it to the AES, one masked AES's S-box over GF(2^8)with two 8-bit input and output masks needs to store 28×28×256bytes(16.8Mbytes). Therefore, for a whole 128-bit masked AES with an unrolled architecture, it needs to store around 2952.8 Mbytes. This is too big to be fit into any field programmable gate array (FPGA). To have a feasible FPGA implementation, one possible way is to transform the S-box computation of a masked AES from GF (2^8) to GF (2^4).

We perform the masked AES mainly over GF (2^4), and the related operations like the masked MixColumns, masked AddRoundKey, and masked ShiftRows including redundant masking values are all calculated over GF (2^4). Therefore, we only need to transform the input values from GF(2^8) to GF(2^4)and transform the output values back from GF(2^4) to GF(2^8) once which reduces around 20.5% hardware resources. In addition, we map half resources of the masked S-box onto block RAM (BRAM) which reduces 15.7%hardware resources. We insert pipelined registers into the round calculation and its masked S-box calculation in order to meet the requirement of high throughput for SANs. We show that the area-optimized design can be fit into Xilinx Virtex-6 plat form, which provides a feasible alternative protection of the AES on reconfigurable devices. We also perform DPA attack to the iterative version of the masked AES, and we prove that no correct bytes of the last round key can be guessed from our masked design.

The rest of this brief is organized as follows. Section II Presents the existing works. Section III proposes the optimized AES with an unrolled architecture, presents the detailed design methodologies about the masked S-box and masked MixColumns, and further optimizes the proposed design by inserting pipelined registers.

II. PREVIOUS WORK

Data transformations in a SAN system usually need real-time high-throughput processing regardless of area overheads. In addition, the security issues of "data-at-rest" in a SAN system require an add-in masking to be DPA and glitch attack resistant. Most existing works only concern high throughputs but not the ability to defend DPA and glitch attacks. Gaj and Chodowiec [5] proposed a pipelined structure for the AES on Virtex XCV-1000 FPGA and achieved 12 Gbits/s. Standaert et *al.* [6] presented the design tradeoff for the further optimization of the AES implementation on FPGA platforms. Unrolling, tiling, and pipelining structures for the AES were discussed in [7]. McLoone and McCanny's method achieved a throughput of 12 Gbits/s using lookup table (LUT)-based SubBytes [8]. Another approach [9] aimed at the on-the-fly generation of SubBytes was first proposed by Rijmen, one of the creators of the AES. Hodjat and Verbauwhede presented a fully pipelined SubBytes architecture achieving a throughput of 21.54 Gbits/s . However, all the afore mentioned methods are vulnerable to DPA and glitch attacks. Mangardet al. successfully broke the AES by using the DPA attack at the algorithmic level. Oswald *et al*. proposed a masked SubBytes over GF (2^4) at the algorithm level, but they only focused on software implementation. Higher order masking schemes have been proposed. They are based on software implementations of the masked AES. The countermeasures used in the work of Goli´ c and Canright and Batina can be attacked successfully by the glitch attack at the gate level. To the best of our knowledge, no previous work has been done on the high-throughput masked AES that has the ability to defend against DPA and glitch attacks. This is due to the required huge hardware area when applying masking to AES at the algorithm level.

III. Proposed masked AES of unrolled architecture

In the Boolean masking implementation, the intermediate Value x is concealed by exclusive-ORing it with the random Mask m. In the round function of the AES, ShiftRows, MixColumns, and AddRoundKey are linear transformations, while SubBytes is the only nonlinear transformation of the AES. However, the masked nonlinear transformation SubBytes has the characteristic as S-box(x $\oplus$ m) not equal to S-box(x) $\oplus$ S-box (m). In order to mask the nonlinear transformation, a new S-box, denoted as S-box_, is recomputed as S-box'(x $\oplus$ m) =S-box(x) $\oplus$ m', where m and m' are the input and output masks of SubBytes. To mask a 128-bit AES, it usually needs 6-byte random values. These 6 values are defined as m, m', m1, m2, m3, and m4. For simplicity, it is defined as m1234= {m1,m2,m3,m4} aand the mask for one 32-bitMixColumns transformation, and it also holds that m'1234 = MixColumns (m1234). The field GF(2^8) is an extension of the field GF(2^4), over which to perform a modular reduction needs an irreducible polynomial of degree 2, x^2 + {1}x + {e}, and another irreducible polynomial of degree 4, x^4 + x+1. In order to reduce the hardware resources, to calculate the masked AES engine mainly over GF (2^4). Fig.3.1 shows the proposed masked AES, which moves the mapping and inverse mapping outside the AES's round functions. The plaintext and the masking values are mapped once from GF (2^8) to GF(2^4), and all the intermediate operations are computed over GF(2^4). Finally, the cipher text is mapped back from GF(2^4) to the original field GF(2^8). In this brief,all the masking values need to be mapped from GF(2^8) to GF(2^4), and denote as m84 = map(m), m'84 = map(m'), m1234,84 = map(m1234), and m'1234,84 = map(m'1234). The adjustment of the masked SubBytes and masked MixColumns is discussed in the following, and the masked ShiftRows and masked AddRoundKey remain the same.

A. *Optimized Masked S-Box over GF (2^4)*

In order to move the mapping and inverse mapping outside AES's round operation, The exchange of computational sequence of masked affine and inverse mapping functions within masked S-box. The masked affine function needs to be adjusted with new scaling factors. In Fig. 1(b), map operation is the mapping transformation of 8 × 8 matrix, and map^{-1} is constructed by the inverse map operation. The input values of the map function are denoted as (z +m) and m, and the output values of the map function are (z + m)' and m',

Where {(z + m), m} ∈ GF (2^8) and {(z + m) _, m_} ∈ GF (2^4).

It holds that (z + m + m)'= map (z + m + m) (1)

Where (z + m) _ = {a*h + mh, a∗l + ml} and m' = {mh, ml}.

As discussed before, maffine and maffine' are needed for scaling the output values and the output masking values. The following

steps introduce the procedure to obtain the scaling values. The normal affine function (Ax+b) can be applied to the left and the right sides of (1) as
A (z + m + m) + b = Amap−1(z + m + m)'+ b. (2)
When mapping Equation 2 from GF(2^8) to GF(2^4), we can getmap (A(z +m + m) + b)= map _Amap−1(z + m + m)'+ b' (3) map (A(z + m) + b)+map Am= mapAmap−1(z + m)'+ mapb + mapAmap−1m'. (4)

Therefore, we deduce that maffine = mapAmap−1 + mapb and Maffine'= mapAmap−1. The Fig. 3.2 (b) shows the new

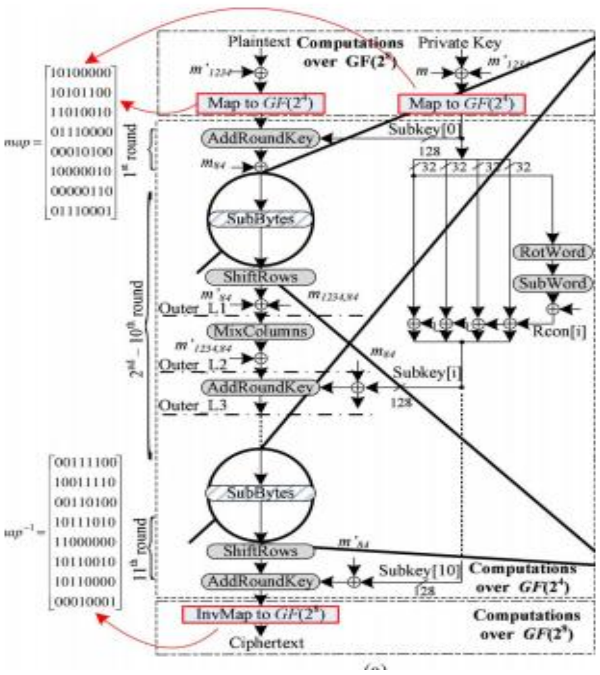
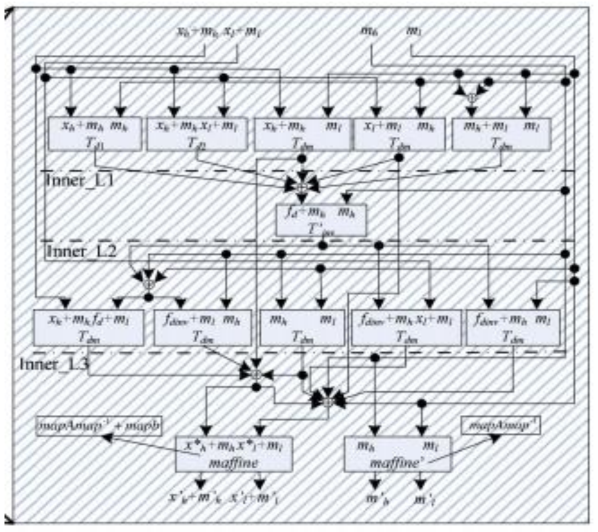


Fig 3.1(a): Masked AES



(b)

Fig 3. 1(b): Masked S-box

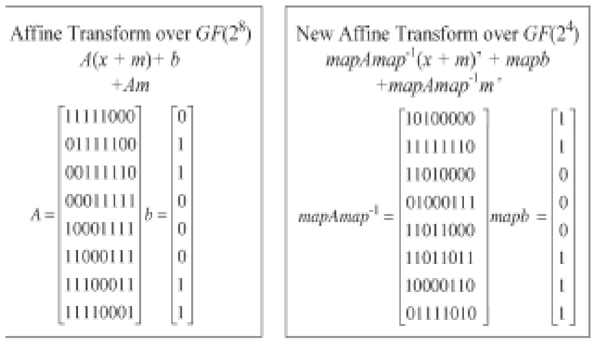


Fig 2: Affine transform function from GF(2^8) to GF(2^4)

B. *PRPOSED MASKED MIXCOLUMNS OVER GF(2^4)*

Masked MixColumns can be scaled to adjust the operations over GF(2^4), and it needs to deduce the scaling factor of a modular multiplication with the fixed coefficients 0X02 and 0X03. If S is 1 byte of MixColumns, it holds that S= map(Sh,Sl) ∼= Shx+Sl, where S ∈ GF(2^8) and Sh,Sl ∈GF(2^4). Therefore, scaling factors 2x+6 and 2x+7 of S equal to (4Sh+2Sl)x + fSh+6Sl) and (5Sh+2Sl) x+(fSh+7Sl). Fig. 3 shows the scaling computation for the masked Mixcolumns.

C. *Optimization for Proposed Architecture*

Usually, throughputs can be significantly improved by inserting pipeline registers for latency careless designs. For each masked AES's round, we insert six-stage pipelines to enhance the throughputs.

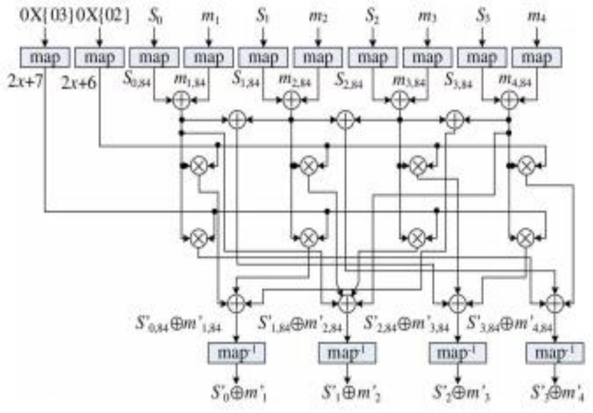

0X{03}0X{02}
S_0 m_1 S_1 m_2 S_2 m_3 S_3 m_4
map
2x+7
2x+6
$S_{0,84}$ $m_{1,84}$ $S_{1,84}$ $m_{2,84}$ $S_{2,84}$ $m_{3,84}$ $S_{3,84}$ $m_{4,84}$
$S'_{0,84}\oplus m'_{1,84}$
$S'_{1,84}\oplus m'_{2,84}$
$S'_{2,84}\oplus m'_{3,84}$
$S'_{3,84}\oplus m'_{4,84}$
map^{-1}
$S'_0\oplus m'_1$
$S'_1\oplus m'_2$
$S'_2\oplus m'_3$
$S'_3\oplus m'_4$

Fig 3: Scaling computation of the masked MixColumns

We insert three pipelines to each round of the masked AES, called outer three pipelines, as shown in Fig. 1(a). The pipeline registers are inserted at the output of each transformation. We insert three pipelines to the masked S-box, called inner three pipelines, as shown in Fig. 1(b). Note that the maximum pipelined stages for our proposed design is six. In order to be compatible with the encryption procedure, we also insert six-stage pipelines to the key expansion in order not to affect the critical path of the main encryption.

IV. HIGH SECURITY PERFORMANCE

To improve security in the AES instead of manually giving the key, we will extract the key from pixel values of the Retina image. So, we can prevent the hacking of key from various attacks. Thus, the high security AES can be introduced. Here, we use matlab for converting the Retina image portions into pixel (digital) vaues. The following block diagram fig :IV depicts the idea.

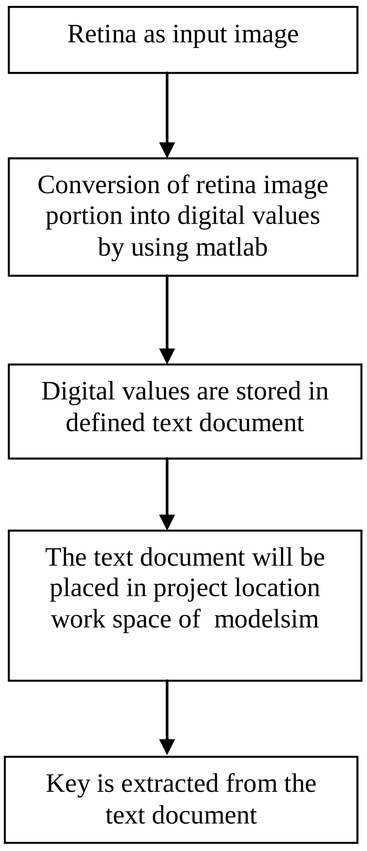
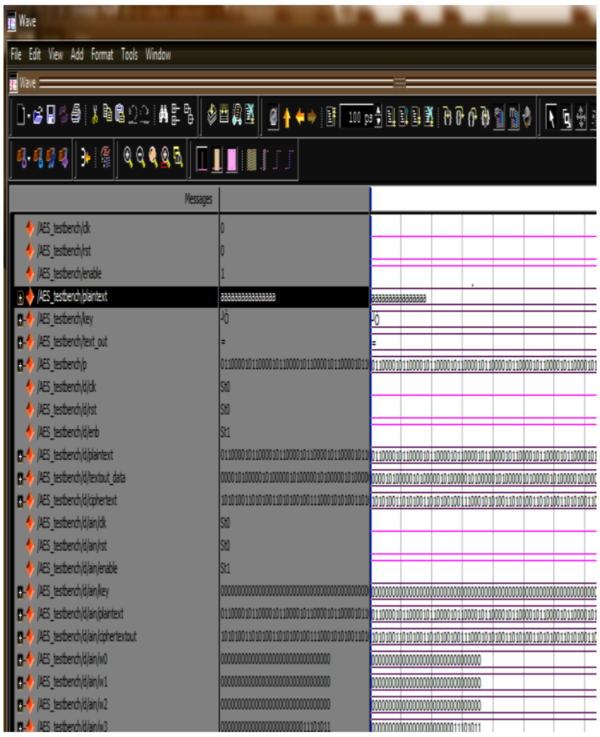

Retina as input image
Conversion of retina image portion into digital values by using matlab
Digital values are stored in defined text document
The text document will be placed in project location work space of modelsim
Key is extracted from the text document

Fig iv: block diagram for key extraction from retina image

V. RESULTS

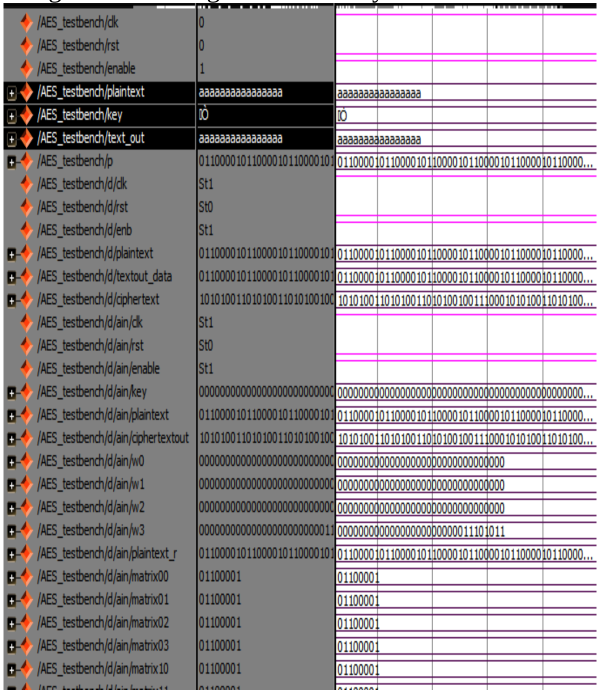
Here, we showed how masking is perfectly done by masking values both on Encryption and Decryption side. The following Fig:a and Fig:b show perfect masking by introducing both wrong and correct masked ouputs.



```
/AES_testbench/clk                 0
/AES_testbench/rst                 0
/AES_testbench/enable              1
/AES_testbench/plaintext           aaaaaaaaaaaaaaaa
/AES_testbench/key                 ¹Ò
/AES_testbench/text_out            =
/AES_testbench/p                   0110000101100001011000010110000101100001011000010110
/AES_testbench/d/clk               St0
/AES_testbench/d/rst               St0
/AES_testbench/d/enb               St1
/AES_testbench/d/plaintext         0110000101100001011000010110000101100001011000010110
/AES_testbench/d/textout_data      00001010000010100000101000001010000010100000
/AES_testbench/d/ciphertext        10101001101010011010100100111000101010011010
/AES_testbench/d/ain/clk           St0
/AES_testbench/d/ain/rst           St0
/AES_testbench/d/ain/enable        St1
/AES_testbench/d/ain/key           000000000000000000000000000000000000
/AES_testbench/d/ain/plaintext     0110000101100001011000010110000101100001011000010110
/AES_testbench/d/ain/ciphertextout 10101001101010011010100100111000101010011010
/AES_testbench/d/ain/w0            00000000000000000000000000000000
/AES_testbench/d/ain/w1            00000000000000000000000000000000
/AES_testbench/d/ain/w2            00000000000000000000000000000000
/AES_testbench/d/ain/w3            00000000000000000000000011101011
```

Fig v(a): Wrong Masked output

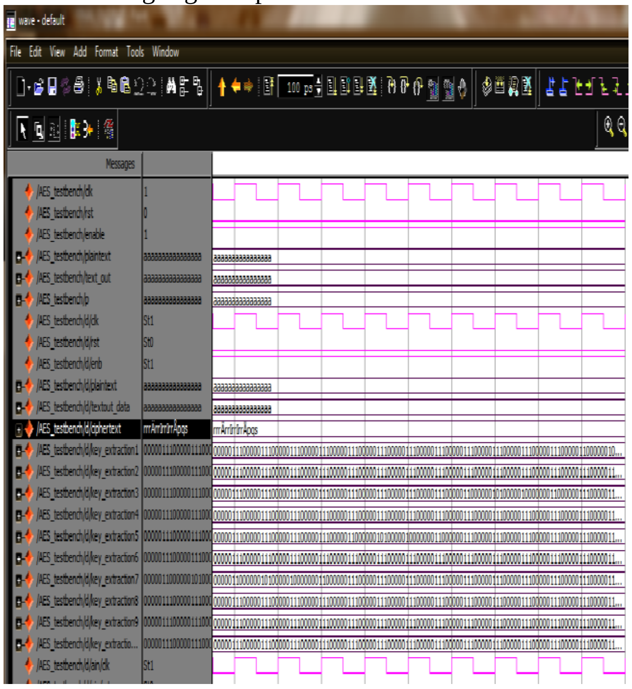
Here, we simulated entire AES encryption and decryption using the same masking values on both side. The plain text can be decrypted at output using same masking values and key.



```
/AES_testbench/clk                  0
/AES_testbench/rst                  0
/AES_testbench/enable               1
/AES_testbench/plaintext            aaaaaaaaaaaaaaaa
/AES_testbench/key                  ÌÒ
/AES_testbench/text_out             aaaaaaaaaaaaaaaa
/AES_testbench/p                    0110000101100001011000010110000...
/AES_testbench/d/clk                St1
/AES_testbench/d/rst                St0
/AES_testbench/d/enb                St1
/AES_testbench/d/plaintext          0110000101100001011000010110000...
/AES_testbench/d/textout_data       0110000101100001011000010110000...
/AES_testbench/d/ciphertext         10101001101010011010100100111000101010011010100...
/AES_testbench/d/ain/clk            St1
/AES_testbench/d/ain/rst            St0
/AES_testbench/d/ain/enable         St1
/AES_testbench/d/ain/key            0000000000000000000000000000000000000000000000...
/AES_testbench/d/ain/plaintext      0110000101100001011000010110000...
/AES_testbench/d/ain/ciphertextout  10101001101010011010100100111000101010011010100...
/AES_testbench/d/ain/w0             00000000000000000000000000000000
/AES_testbench/d/ain/w1             00000000000000000000000000000000
/AES_testbench/d/ain/w2             00000000000000000000000000000000
/AES_testbench/d/ain/w3             00000000000000000000000011101011
/AES_testbench/d/ain/plaintext_r    0110000101100001011000010110000...
/AES_testbench/d/ain/matrix00       01100001
/AES_testbench/d/ain/matrix01       01100001
/AES_testbench/d/ain/matrix02       01100001
/AES_testbench/d/ain/matrix03       01100001
/AES_testbench/d/ain/matrix10       01100001
```

Fig v(b): Correct Masked output

Here, we use retina image for the key to improve high security for the Masked AES more better. We extracted around 10 keys from the test document and we used key_extraction1 as key. The following Fig c: depicts the idea.



```
/AES_testbench/clk                   1
/AES_testbench/rst                   0
/AES_testbench/enable                1
/AES_testbench/plaintext             aaaaaaaaaaaaaaaa
/AES_testbench/text_out              aaaaaaaaaaaaaaaa
/AES_testbench/p                     aaaaaaaaaaaaaaaa
/AES_testbench/d/clk                 St1
/AES_testbench/d/rst                 St0
/AES_testbench/d/enb                 St1
/AES_testbench/d/plaintext           aaaaaaaaaaaaaaaa
/AES_testbench/d/textout_data        aaaaaaaaaaaaaaaa
/AES_testbench/d/ciphertext          rrrÄrrÌrrÌrrÅpqs
/AES_testbench/d/key_extraction1     00000111000001110000
/AES_testbench/d/key_extraction2     00000111000001110000
/AES_testbench/d/key_extraction3     00000111000001110000
/AES_testbench/d/key_extraction4     00000111000001110000
/AES_testbench/d/key_extraction5     00000111000001110000
/AES_testbench/d/key_extraction6     00000111000001110000
/AES_testbench/d/key_extraction7     00000110000001010000
/AES_testbench/d/key_extraction8     00000111000001110000
/AES_testbench/d/key_extraction9     00000111000001110000
/AES_testbench/d/key_extractio...    00000111000001110000
/AES_testbench/d/ain/clk             St1
```

Fig v(c): Masked AES ouput using Retina image for Key extraction

VI. CONCLUSION

High security is an important factor for encryption algorithms in cryptography. The proposed masked AES only needs to map the plaintext and masking values from GF (2^8) to GF(2^4) once at the beginning of the operation and map the cipher text back from GF(2^4) to GF(2^8) once at the end of the operation. Therefore, by moving the mapping and inverse mapping outside the masked AES's round function. In addition to this here, we use retina image as a key which results in further development in encryption standards.

REFERENCES

[1] Advanced Encryption Standard (AES), FIPS-197, Nat. Inst. of Standards and Technol., 2001.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in Proc. CRYPTO, 1999, vol. LNCS 1666, pp. 388 397.

[3] L. Goubin and J. Patarin, "DES and differential power analysis (the 'duplication' method)," in Proc. CHES LNCS, 1999, vol. 1717, pp. 158–172.

[4] S. Messerges, "Securing the AES finalists against power analysis attacks," in Proc. FSE LNCS, 2000, vol. 1978, pp. 150–164

[5] A. Jaya Lakshmi, I. Ramesh Babu, "Design of security key Generation algorithm using Fingerprint based Biometric Modality", ISSN: 2250-3021, Vol 2, Feb 2012.

[6] P. Balakumar, R. Venkatesan, "A Survey on Biometric Based Cryptographic key Generation Scheme", ISSN 2249 - 9555, Vol 2, No1, 2012.

[7] J. D. Goli´ c, "Techniques for random masking in hardware," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 54, no. 2, pp. 291–300, Feb. 2007

[8] V. Rijmen, "Efficient Implementation of the Rijndael S-Box," Dept. ESAT., Katholieke Universiteit Leuven, Leuven, Belgium, 2006. [Online]. Available: http://www.networkdls.com/Articles/sbox.pdf

[9] A. Hodjat and I. Verbauwhede, "A 21.54 Gbits/s fully pipelined processor on FPGA," inProc. IEEE 12th Annu. Symp. Field-Programm. Custom Comput. Mach., 2004, pp. 308–309