International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 2; Issue: 4; April-2014

PERFORMANCE EVALUATION OF FPGA BASED MASKED AES TECHNIQUES FOR STORAGE AREA NETWORK

Navin S. M
4th Sem M.Tech Student, Dept. of ECE
BNMIT, Bangalore-560070
navsm007@gmail.com

Dr. P. A. Vijaya
Professor, Dept of ECE
BNMIT, Bangalore-560070
pavmkv@gmail.com

***ABSTRACT*-Storage area networks are used in government and industry to store large amount of data while assuring availability and access to that data and security. 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 usually adopts the unrolling technique which requires extremely large field programmable gate array (FPGA) resources, high computational time. This research work aims to optimize the area with reduce computation time and high throughput for a masked AES with an unrolled structure. This is achieved by mapping its operations from GF(2^8) to GF(2^4) as much as possible. The number of mapping GF(2^8) to GF(2^4) and inverse mapping GF(2^8) operations of the masked SubBytes step are reduced from ten to one. In order to be compatible, the masked MixColumns, masked AddRoundKey, and masked ShiftRows including the redundant masking values are carried over GF(2^4). Masking and mapping techniques increases computational time which leads to usage of pipelining techniques to reduce the computational time. A FPGA block RAM (BRAM) is used to further reduce hardware resources.**

Keywords: Advanced Encryption Standards (AES), Differential Power Analysis (DPA), Field Programmable Gate Array (FPGA), Masking, Storage Area Network (SAN).

1. INTRODUCTION

Securing data from accidental or malicious disclosure, whether it is data-in-transit or data-at-rest, is critical to the mission of any organization. SAN security should be carefully considered and then implemented in accord with all applicable security policies. One important type of SAN is the Fibre Channel SAN used for the rapid transfer of data between servers and FC storage devices via FC switches. Information should be cryptographically protected in SANs when it is at rest on a FC storage devices, To protect data-in-transit encrypt data-in-transit and all communication between FC devices. To protect data-at-rest the data should be encrypted before arriving at its destined storage device. This requires the use of special purpose appliances that can encrypt the data that is being sent to a storage device. These applications need not only the protection at both the protocol level and the physical level but also high-throughput implementation. For Example, it needs upto 40 Gbit/s throughput for a four port host bus adapter connected by fiber cables. The information leakage includes power consumption, timing and fault detection.

In 1999, Kocher et al. first broke the normal advanced encryption standard (AES) 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 or nonstandard CMOS cells. 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. 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-box over GF(2^8) with two 8-bit input and output mask needs to store GF(2^8) to GF(2^4). Therefore, for a whole 128-bit masked AES with unrolled architecture, it needs to store around 2952.8 Mbytes. This is too big to be fit into any FPGA.

In this paper, we develop mapping technique to perform the masked AES mainly over GF(2^4). In addition we map masked S-box onto block RAM which reduces area. We use pipelining technique in order to meet the throughput of high throughput for SANs. We also perform masked AES in different modes of operation. Finally we compare all the techniques used to develop masked AES and conclude accordingly.

2. LITERATURE SURVEY

To carry out this research work, a thorough literature survey was done. It has been briefly explained in this section. In this paper "Efficient implementation of Rijndael encryption in reconfigurable hardware: Improvements and design tradeoffs," 2003[6], performance evaluation of the Advanced Encryption Standard candidates has led to intensive study of both hardware and software implementations. However, although plentiful papers present various implementation results, it seems that efficiency could still be greatly improved by applying good design rules adapted to devices and algorithms. This paper addresses various approaches for efficient FPGA implementations of the Advanced Encryption Standard algorithm. As different applications of the AES algorithm may require different speed/area tradeoffs, we propose a rigorous study of the possible implementation schemes, but also discuss design methodology and algorithmic optimization in order to improve previously reported results. In the paper "A 21.54 Gbits/s fully pipelined processor on FPGA," 2004[10] presents the architecture of a fully pipelined AES encryption processor

on a single chip FPGA. By using loop unrolling and inner-round and outer-round pipelining techniques, a maximum throughput of 21.54 Gbits/s is achieved. A fast and an area efficient composite field implementation of the byte substitution phase is designed using an optimum number of pipeline stages for FPGA implementation. A 21.54 Gbits/s throughput is achieved using 84 block RAMs and 5177 slices of a VirtexII-Pro FPGA with a latency of 31 cycles and throughput per area rate of 4.2 Mbps/Slice. [11] "Successfully attacking masked AES hardware implementations," 2005. During the last years, several masking schemes for AES have been proposed to secure hardware implementations against DPA attacks. In order to investigate the effectiveness of these countermeasures in practice, we have designed and manufactured an ASIC. The chip features an unmasked and two masked. It turns out that masking the AES S-Boxes does not prevent DPA attacks, if glitches occur in the circuit. In the paper "A side-channel analysis resistant description of the AES S-box," 2005[12] presents efficient algorithmic countermeasures to secure the AES algorithm against (first-order) differential side-channel attacks which has been very expensive to implement. In this article, we introduce a new masking countermeasure which is not only secure against first-order side-channel attacks, but which also leads to relatively small implementations compared to other masking schemes implemented in dedicated hardware. Our approach is based on shifting the computation of the finite field inversion in the AES S-box down to GF(2^4).

3. EXISTING METHODS

3.1 *AES*

AES has been adopted by the U.S. government and is now used worldwide. It supersedes the Data Encryption Standard (DES), which was published in 1977. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data. AES is based on a design principle known as a substitution-permutation network, and is fast in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network. The basic information unit for treatment in the AES algorithm is a series of eight bits processes considered as a single unit. The bit series corresponding to the input, the output and the cipher key are processed as arrays of bytes; called State. The State array consists of four columns of bytes, and every column contains 4 bytes. The AES algorithm operates in rounds and support three different key lengths, 128, 192, and 256 bits; the standard will consider only 128-bit as legal block length. The number of these rounds is chosen depending on the key size. In fact, for a key length equal to 128, 192 or 265 the number of rounds is equal to 10, 12 and 14, respectively. i.e.,10 cycles of repetition for 128-bit keys, 12 cycles of repetition for 192-bit keys, 14 cycles of repetition for 256-bit keys.

3.2. Masked AES WITH 6 PIPELINE STAGES

In a masked AES, a random mask is added to the input data of the algorithm prior to encryption. At the end of the encryption, the mask is removed to get the correct result. In order to remove the mask, we need to keep track how the mask is modified by the algorithm. Figure 2.1 shows this basic principle add random mask remove random mask Masked Algorithm Plaintext Cipher text Mask Modification Random Mask Cipher Key Figure 2.1. Basic masking principle masking the linear AES functions is easy. Because these functions are linear, applying them on a masked value A+ X, gives the same result as applying them first on the data A and then on the mask X: f (A+X) = f (A)+ f (X).

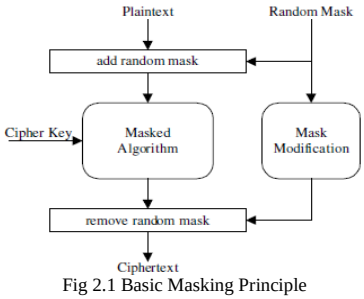


Fig 2.1 Basic Masking Principle

The SubBytes transformation is composed of a multiplicative inversion in GF(2^8) and an affine transformation. Masking the non-linear byte inversion is tricky because Inv(A+X) ≠ Inv(A)+Inv(X). Without modification, the result of the byte inversion is $(A + X)^{-1}$ and thus it is not possible to remove the mask at the end of the algorithm easily. Therefore, a modified byte inversion is required such that the result of the inversion equals A^{-1} +X. Multiplicative masking, which was presented by Akkar et al., is based on the idea that prior to the byte inversion the additive mask X is replaced by the multiplicative mask Y. After the byte inversion, the multiplicative mask is replaced by the additive mask again. As it can be seen in Figure 2.2, SubBytesAkkar requires four multipliers, two inversions and two additions in GF(2^8). Therefore, it is already clear that masking leads to a noticeable increase in terms of area viewed as a multiplication by a particular MDS matrix in a finite field.

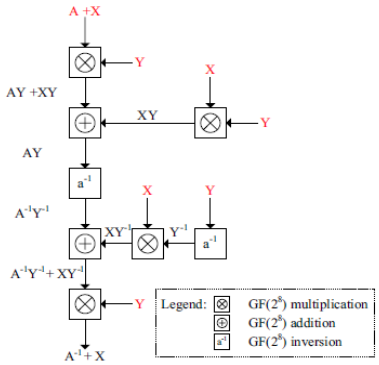


Fig 2.2 Modified Byte Inversion *SubByteAkkar*[7]

The Boolean masking is a good candidate to be applied to the AES in SANs, but if it is directly applied 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 × 256 bytes (16.8 Mbytes). Therefore, for a whole 128-bit masked AES with an unrolled architecture, it needs to store around 2952.8Mbytes. 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).

Techniques to optimize the area of a masked AES for SANs are developed and the masked AES mainly over GF(2^4) are performed, and the related operations like the masked MixColumns, masked AddRoundKey, and masked ShiftRows including redundant masking values are all calculated over GF(24). Therefore, there is 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.

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. We define the linear transformations as Oper; then, the masked Oper can be written as Oper(x ⊕ m)=Oper(x) ⊕ Oper(m). However, the masked nonlinear transformation SubBytes has the characteristic as S-box(x ⊕ m)' = S-box(x) ⊕ S-box(m). In order to mask the nonlinear transformation, a new S-box, denoted as S-box', is recomputed as S-box (x ⊕ m) = S-box(x) ⊕ m, where m and m' are the input and output masks of SubBytes. Usually, throughputs can be significantly improved by inserting pipeline registers for latency careless designs. For each masked AES's round, six-stage pipelines are inserted to enhance the throughputs. Three pipelines to each round of the masked AES, called outer three pipelines, are inserted as shown in Fig. 1. The pipeline registers are inserted at the output of each transformation. Note that the maximum pipelined stages for our proposed design is six. In order to be compatible with the encryption procedure, six-stage pipelines are inserted to the key expansion in order not to affect the critical path of the main encryption.

4. PROPOSED MASKED AES METHODS WITH UNROLLED ARCHITECTURE & PIPELINING TECHNIQUE

4.1 Masked AES with reduced computation time

Techniques have been generated to reduce the computation time using 10 pipeline stages with the trade-off with area. Using 10 pipeline stages greatly reduces the computation time but the area is increased as compared to 6 pipeline stage technique. Another method is to reduce the shift row operation and mix column without any effects on security.

4.2. Masked AES in CBC and PCBC mode

CIPHER-BLOCK CHAINING (CBC)

In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block. If the first block has index 1, the mathematical formula for CBC encryption is

$$C_i = E_k (P_i + C_{i-1}), C_0 = IV$$

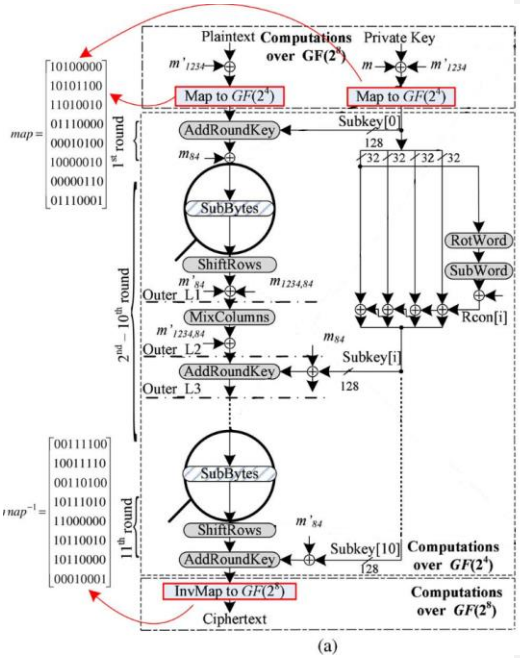


Fig.1 Proposed masked AES with an unrolled architecture of Masked AES with pipelining technique

while the mathematical formula for CBC decryption is

$$P_i = D_k (C_i) + C_{i-1}, C_0 = IV$$

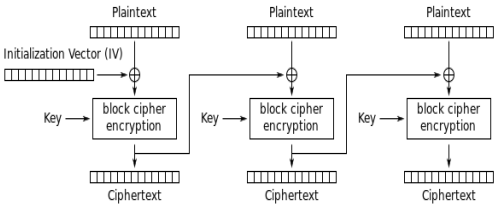


Fig. 4.1 Cipher block chaining mode encryption [7]

CBC has been the most commonly used mode of operation. Its main drawbacks are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size. One way to handle this last issue is through the method known as cipher text stealing. Note

that a one-bit change in a plaintext or IV affects all following cipher text blocks. Decrypting with the incorrect IV causes the first block of plaintext to be corrupt but subsequent plaintext blocks will be correct. This is because a plaintext block can be recovered from two adjacent blocks of cipher text. As a consequence, decryption *can* be parallelized. Note that a one-bit change to the cipher text causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext, but the rest of the blocks remain intact.

PROPAGATING CIPHER-BLOCK CHAINING (PCBC)

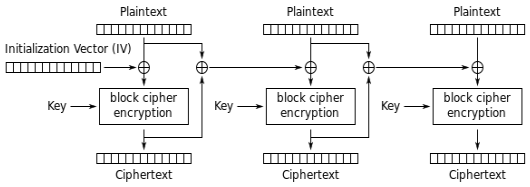


Fig. 4.2 Propagating Cipher block chaining mode encryption [7]

The propagating cipher-block chaining or plaintext cipher-block chaining mode was designed to cause small changes in the ciphertext to propagate indefinitely when decrypting, as well as when encrypting.

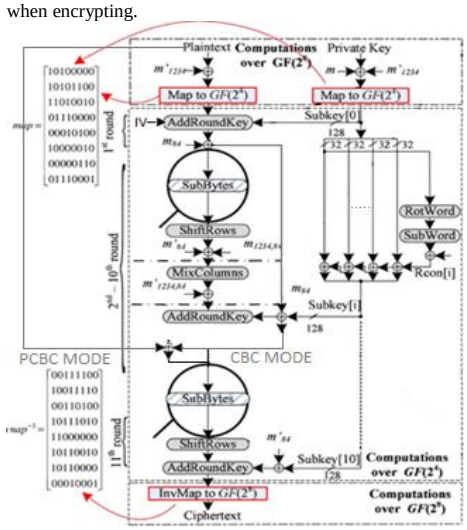


Fig. 4 Proposed masked AES with an unrolled architecture in CBC mode and PCBC mode

4.3 Masked AES in CBF and OBF mode

CIPHER FEEDBACK MODE (CFB)

The cipher feedback (CFB) mode, a close relative of CBC, makes a block cipher into a self-synchronizing stream cipher. Operation is very similar; in particular, CFB decryption is almost identical to CBC encryption performed in reverse:

$$C_i = E_k(C_{i-1}) \oplus P_i,\ P_i = E_k(C_{i-1}) \oplus C_i,\ C_0 = IV$$

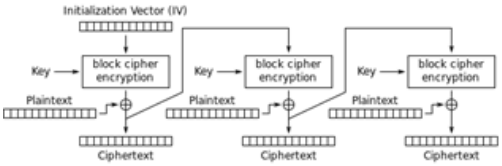


Fig 4.3 Cipher Feedback Mode Encryption

This simplest way of using CFB described above is not any more self-synchronizing than other cipher modes like CBC. If a whole blocksize of ciphertext is lost both CBC and CFB will synchronize, but losing only a single byte or bit will permanently throw off decryption. To be able to synchronize after the loss of only a single byte or bit, a single byte or bit must be encrypted at a time. CFB can be used this way when combined with a shift register as the input for the block cipher. To use CFB to make a self-synchronizing stream cipher that will synchronize for any multiple of x bits lost, start by initializing a shift register the size of the block size with the initialization vector. This is encrypted with the block cipher, and the highest x bits of the result are XORed with x bits of the plaintext to produce x bits of ciphertext. These x bits of output are shifted into the shift register, and the process repeats with the next x bits of plaintext. Decryption is similar, start with the initialization vector, encrypt, and XOR the high bits of the result with x bits of the ciphertext to produce x bits of plaintext. Then shift the x bits of the ciphertext into the shift register. This way of proceeding is known as CFB-8 or CFB-1 (according to the size of the shifting). If x bits are lost from the ciphertext, the cipher will output incorrect plaintext until the shift register once again equals a state it held while encrypting, at which point the cipher has resynchronized. This will result in at most one blocksize of output being garbled. Like CBC mode, changes in the plaintext propagate forever in the ciphertext, and encryption cannot be parallelized. Also like CBC, decryption can be parallelized. When decrypting, a one-bit change in the ciphertext affects two plaintext blocks: a one-bit change in the corresponding plaintext block, and complete corruption of the following plaintext block. Later plaintext blocks are decrypted normally. CFB shares two advantages over CBC mode with the stream cipher modes OFB and CTR: the block cipher is only ever used in the encrypting direction, and the message does not need to be padded to a multiple of the cipher block size (though ciphertext stealing can also be used to make padding unnecessary).

OUTPUT FEEDBACK MODE (OFB)

The output feedback (OFB) mode makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext. Just as with other stream ciphers, flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same

location. This property allows many error correcting codes to function normally even when applied before encryption. Because of the symmetry of the XOR operation, encryption and decryption are exactly the same:

$C_j = P_j + O_j , P_j = Cj + O_j , O_j = E_k (I_j)$

$I_j = O_{j-1} , I_0 = IV$

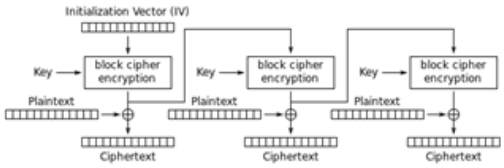
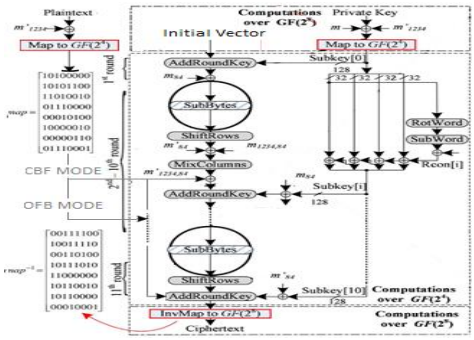


Fig 4.4 Output Feedback Mode Encryption

Each output feedback block cipher operation depends on all previous ones, and so cannot be performed in parallel. However, because the plaintext or ciphertext is only used for the final XOR, the block cipher operations may be performed in advance, allowing the final step to be performed in parallel once the plaintext or ciphertext is available. It is possible to obtain an OFB mode keystream by using CBC mode with a constant string of zeroes as input. This can be useful, because it allows the usage of fast hardware implementations of CBC mode for OFB mode encryption. Using OFB mode with a partial block as feedback like CFB mode reduces the average cycle length by a factor of 2^{32} or more.

5. PROPOSED MASKED AES FOR UNROLLED STRUCTURE

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, Shift Rows, Mix- Columns, and AddRoundKey are linear transformations, while Sub Bytes is the



. Fig. 4 Proposed masked AES with an unrolled architecture in CBF mode and OBF mode

only nonlinear transformation of the AES. The linear transformations are defined as Oper; then, the masked Oper can be written as Oper($x \oplus m$) = Oper(x) $\oplus$ Oper(m). However, the masked nonlinear transformation Sub Bytes has the characteristic as S-box($x \oplus m$) $\neq$ 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 Sub Bytes. 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, m_{1234} = {$m1$, $m2$,$m3$,$m4$} is defined as the mask for one 32-bitMixColumns transformation, and it also holds that m'_{1234}= MixColumns(m_{1234}). 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, the masked AES engine is calculated mainly over $GF(2^4)$. Fig. 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 ciphertext is mapped back from $GF(2^4)$ to the original field $GF(2^8)$. All the masking values need to be mapped from $GF(2^8)$ to $GF(2^4)$, and m_{84} = map(m), m'_{84} = map(m'), $m_{1234,84}$ = map(m_{1234}), and $m'_{1234,84}$ = map(m'_{1234}).

5.1 Optimized Masked S-Box over GF (2^4)

In order to move the mapping and inverse mapping outside AES's round operation, we exchange the 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 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 ($z + m$) and m, and the output values of the map function are ($z + m$)*'* and m', where {($z + m$),m} ϵ $GF(2^8)$ and {($z + m$)_,m_} ϵ $GF(24)$. It holds that

$$(z + m + m)' = \text{map}\ (z + m + m) \quad (1)$$

where ($z + m$)*'* = {$a_h^* + mh$,$a_l^* + ml$} and m' = {m_h,m_l}. 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

$$A(z+m+m) + B = A\ \text{map}'\ (z+m+m)' + B \quad (2)$$

When mapping Equation (2) from $GF(2^8)$ to $GF(2^4)$,

$$\text{map}\ (A(z+m+m) + B) = \text{map}\ (A\ \text{map}'\ (z+m+m)' + B) \quad (3)$$

$$\text{map}\ (A(z+m) + B) + \text{map}\ Am = \text{map}\ A\ \text{map}'\ (z+m+)' + \text{map}\ B + \text{map}\ A\ \text{map}'\ m' \quad (4)$$

Therefore we deduce that ,

Maffine = map A map' (z+m+)' + map B + map A map' m'
Therefore, maffine = $mapAmap^{-1}$ + map$b$ and maffine'= $mapAmap^{-1}$.

6. RESULT

In this section, I have implemented the proposed design and synthesized the design using Xilinx ISE 13.1 Virtex-6 XC6VLX240T platform. The table shows the comparison between different AES techniques for storage of data in network. In order to have fair comparison, I also implement the masked AES design using Oswalds masked S-Box with non-pipelined and 6- stages pipelined unrolled structure and basic iterative unprotective AES method. The.proposed masked AES is implemented in CBC and PCBC mode, CBF and OFB to achieve further more security.

7. CONCLUSION

High throughput is an important factor for large data transformation systems in SANs. In this brief, an LUT-based masked S-box has been proposed to construct the DPA-resistant design with acceptable area on FPGA. 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 ciphertext 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, we can reduce area resources. We also map some parts of the masked S-box onto BRAM which further reduces area resources. We also reduce shift rows operation to increase the computational time without any effect to the security. We achieve 2.953Gbits/s throughput for the proposed masked AES. Finally, the proposed masked AES is implemented in CBC mode to achieve more security.

8. 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] S. Messerges, "Securing the AES finalists against power analysis attacks," in Proc. FSE LNCS, 2000, vol. 1978, pp. 150–
[4] Yi Wang and Yajun Ha, "FPGA-Based 40.9-Gbits/s Masked AES with Area Optimization for Storage Area Network," IEEE Transaction on Circuits and System-II, 2013.
[5]http://en.wikipedia.org/wiki/Block_cipher_mode_of_operatio
[6] M. McLoone and J. V. McCanny, "Rijndael FPGA implementations utilizing look-up tables," in Proc. IEEE Workshop Signal Process. Syst., Antwerp, Belgium, 2001, pp. 349–360.
[7] V. Rijmen, "Efficient Implementation of the Rijndael S-Box," Dept. ESAT., Katholieke Universiteit Leuven, Leuven, Belgium, 2006.
[8] Yi Wang and Yajun Ha " FPGA Based Masked AES with Area Optimization for Storage Area Network," 2013.

TABLE 1. RESULTS OF DIFFERENT AES

METHODS	TECHNIQUES	AREA (SLICES)	Computational Time	Throughput (Gbit/sec)	Security
EXISTING METHODS	Unprotected AES	6985	100ns	1.28	LOW
	Masked AES	13677	240ns	0.533	HIGH
	Masked AES with Pipelining	10345	130ns	2.953	HIGH
PROPOSED METHODS	Masked AES in CBC mode	12678	150ns	2.56	VERY HIGH
	Masked AES in PCBC mode	12451	150ns	2.56	VERY HIGH
	Masked AES in CBF mode	11456	140ns	0.914	VERY HIGH
	Masked AES in OFB mode	10967	140ns	0.914	VERY HIGH

Comment [NS2]:

Comment [NS1]:

AUTHORS PROFILE

Mr. Navin S.M is doing M.Tech in the Dept. of E&C Engg, BNMIT, Bangalore, Karnataka, India. This paper is based on the project work done by him under the guidance of Dr. P. A. Vijaya.

Dr. P.A Vijaya did her B.E. from MCE, Hassan and M.E. and Ph.D. from IISc, Bangalore. She worked in MCE, Hassan, Karnataka for about 27 years. Presently, she is a Professor in the Dept. of E&C Eng., BNMIT, Bangalore, Karnataka, India. Two students have obtained Ph.D. degree under her guidance and four more are doing Ph.D. Her research interests are in the areas of Pattern Recognition, Image Processing, Embedded Systems, Real time Systems, Network Security and Operating Systems.