IPHV7I10022X
International Journal Of Advanced Research and Innovation -Vol.7, Issue .I
ISSN Online: 2319 – 9253
Print: 2319 – 9245

IMPLEMENTATION OF PSEUDO-RANDOM SEQUENCE GENERATOR (PRNG) BASED ON SECURE HASH -1 ALGORITHM

[1] CH .S.RANADHEER, [2] RAMESH JITTY

Dept of ECE,
TRINITY COLLEGE OF ENGINEERING AND TECHNOLOGY, KARIMNAGAR.

Abstract: - A novel stream cipher based on the popular linear feedback shift register (LFSR) with a dynamic feedback network controlled by a decoder circuit is presented. The proposed circuit can be employed efficiently in cryptographic applications owing to its improved inviolability property compared to traditional LFSRs. Indeed, although the proposed topology and traditional LFSRs have similar statistical properties, it exhibits a very high linear complexity (LC) and the sequences generated from the novel topology are much more difficult to be predicted. The proposed cipher is described and design criteria to choose the key of the system are also given. As an example, a 16-bit length generator was implemented on a Xilinx FPGA and experimentally verified using the most common statistical and randomness tests. Then it was also designed in standard cell AMS technology. Moreover, to test the generator's inviolability, a novel methodology, based on a multiperceptron neural network, was also developed and is also presented. Secure Hash standard specifies algorithm that can be used to generate digests of messages. The digests are used to detect whether messages have been changed since the digests were generated.

Keywords: - *Computer Security, Cryptography, Message Digest, Hash Function, Hash Algorithm, Federal Information Processing Standards, Secure Hash Standard.*

I.INTRODUCTION

Today, random bit generators and pseudo-random bit generators are used widely in much electronic equipment in areas such as entertainment, music, simulation and testing. Moreover, these generators are becoming particularly useful to ensure secure data transmission over an insecure communication channels.

From a mathematical point of view, the randomness concept is defined as an independent sequence of numbers which have a specific distribution and probability Random number generators (RNG) can be implemented easily using a circuit or by means of algorithm execution (software) The effective randomness of the hardware solutions depends on physical characteristics such as thermal noise, radioactive decay, high frequency jitter or chaotic behavior.

In contrast, software implementations of RNGs are based on an algorithm that gives an output sequence with a constant statistical distribution and a low auto and mutual correlation. Since the algorithm and the initial condition (seed) are known, these generators are considered deterministic. For this reason they are called pseudo-random number generators (PRNGs).

A good PRNG is characterized by the following properties:

Repeatability: It gives the same output sequence when the same seed is used;

Randomness. It passes most common standard randomness tests and has good statistical properties;

'Implementation of pseudo-random sequence generator based on sha1 algorithm' presents a new and fast random generator which is based on shift-register sequences a pseudo-random mode.

IPHV7I10022X
International Journal Of Advanced Research and Innovation -Vol.7, Issue .I
ISSN Online: 2319 – 9253
Print: 2319 – 9245

The main objective of this is to design and implementation of a implementation of pseudo-random sequence generator (prng) based on secure hash algorithm1. The algorithm iterative, one-way hash functions that can process a message to produce a condensed representation called a message digest. This algorithm enables the determination of a message's integrity: any change to the message will, with a very high probability, result in a different message digests. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers or bits.

II. RELATED WORK

Random numbers are used in a wide variety of applications. Although a majority of random number generators have been implemented in software level, increasing demand exists for hardware implementation due to the advent of faster and high density Field Programmable Gate Arrays (FPGA). FPGAs make it possible to implement complex systems, such as numerical calculations, genetic programs, simulation algorithms etc., at hardware level. This paper discusses in detail the hardware implementation of several RNGs and their characteristics. Somewhat complex Cellular Automata based RNGs show slightly improved performance compared to the simplest Linear Feedback Shift Register RNG

Random numbers are widely used in various applications such as Monte Carlo simulations, cryptography, simulations of wireless communication systems, electronic circuit testing, genetic programming, data encryption, games etc. Usually, random numbers are generated using software algorithms True random numbers can be generated from a physical process, such as measuring thermal noise or noise power level in a radio-frequency receiver, photoelectric effect or other quantum phenomena. These processes are, in theory, completely unpredictable. True random number generators can be implemented by combining both analog and digital electronics. These generators generally tend to be expensive as well as slow.

First, the implementation of most simple and common random number generator, Linear Feedback Shift Register (LFSR) is This generator tends to fail basic requirement of a RNG due to high correlation in the sequence . The correlation problem that can be reduced by modifying the LFSR random number generator is discussed next. Then a cellular automata based random number generator is discussed.

The target hardware for this work was XC4005XL Field Programmable Gate Array (FPGA) on the XS40 prototype board running at 50 MHz. The basic building block of the FPGA is Configurable Logic Block (CLB). It contains two, four-input function generators (F and G) and one three-input function generator (H). These function generators are capable of implementing any arbitrarily defined Boolean function of four inputs and three inputs [4]. They are implemented as memory look up tables. The total number of user configurable logic gates available in this device is 5000.

8-bit LFSR RNG

The most common way to implement a random number generator is LFSR. Codes generated by the LFSR are actually pseudo random sequences because the sequence repeats itself after a certain number of cycles. It is known as the period of the generator. LFSR is based on the recurrence equation,

$$x_n = x_n \oplus x_{n-1} \oplus x_{n-2} \ldots\ldots = \oplus x_{n-m} \quad \ldots\ldots\ldots\ldots\ldots (1)$$

The operator $\oplus$ is the exclusive OR (XOR) operator. The equation shows that *n*th bit can be generated utilizing *m* previous values with XOR operators [2]. The value of *m* determines the period of the generator. The achievable maximum period is 2m-1. For the 8-bit LFSR, the recurrence equation is,

$$x_n = x_{n-2} \oplus x_{n-3} \oplus x_{n-4} \oplus x_{n-8} \quad \ldots\ldots\ldots\ldots\ldots (2)$$

Since new value *x*n depends on previous *m* values, it is necessary to store previous *m* values to find the new value. This can be done with *m* single bit shift registers comprised with flip flops.

According to the equation (2), XOR feedback tap positions are taken at 0^{th}, 4th, 5^{th} and

IPHV7I10022X
International Journal Of Advanced Research and Innovation -Vol.7, Issue .I
ISSN Online: 2319 – 9253
Print: 2319 – 9245

6th flip-flops. The maximum period of the generator is 2^8-1 (255). In each clock pulse, generated new bit is inserted to the shift register while the oldest bit shifts out. Output of the 8 flip-flops form the 8-bit random number. A group of flip-flops connected in series are used with XOR gates to construct the LFSR random number generator (see figure 1).

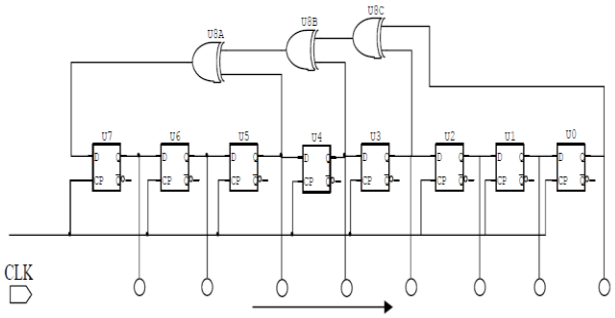

Figure 1: 8-bit LFSR RNG

The recurrence equation depends on the number of bits. Table 1 shows recurrence equations for bits 2 to 8.

Table 1: Taps for Maximal-Cycle LFSRs with 2 to 8 bits

No. of bits	Tap positions	Maximum Period
2	[1,0]	3
3	[2,0]	7
4	[3,0]	15
5	[3,0]	31
6	[5,0]	63
7	[6,0]	127
8	[6,5,4,0]	255

Hardware Implementation of the 8-bit LFSR was straightforward. The LFSR contained 8 D-type flip-flops and 3-two-input XOR gates. The behavior of the circuit was described using the VHDL and the XC4005XL was configured using Web pack software available from Xilinx. It consumed one FG Function generator out of 392 available in the XC4005XL and 8 CLB flip-flops out of 392.

To test the speed, another subsystem was created which comprised a counter (0 to 999). In each clock pulse the counter incremented by one as a new random number was generated. The system gave a pulse when the counter reaches to 999. The time duration between two pulses equal to the generation of 1000 random numbers. The average time taken to generate a random number was about 21.8 ns.

A sample of random numbers generated by LFSR is shown in figure 2. The obvious weakness of the LFSR is the correlation in the sequence. At any time step *t* there is 50% probability that the value at time *t+1* can be predicted. For an n-bit LFSR, if the present value is *v*, then the next value will be $v/2$ or $v/2+2^{n-1}$ [3]. This is shown in figure 3. In LFSR generator, a new random value contains (*m-1*) bits from the old value and only one bit of new information appears in the new value. This could be the reason for high correlation.

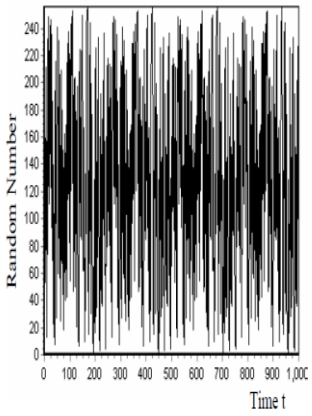

Figure 2: Sample of Random No's

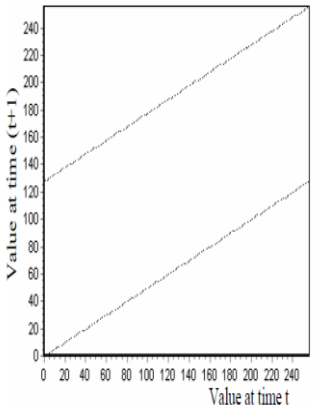

Figure 3: Correlation

However, the random number sequence is highly uniform (see figure 4). Thus, this method is still used in many applications with large registers. As the register size increases, the period becomes extremely large. For instance, if 64 bit register is taken, the period is 18,446,744,073,709,551,615.

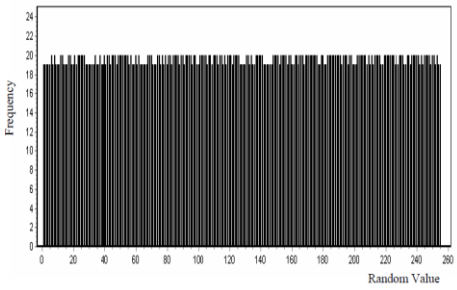

Figure 4: Distribution Random No's

As the seed, a non-zero value should be given to the register before starting the process. If all zero value appears in the register, XOR operations continue to produce zeros and output becomes always zero.

IPHV7I10022X
International Journal Of Advanced Research and Innovation -Vol.7, Issue .I
ISSN Online: 2319 – 9253
Print: 2319 – 9245

III. IMPLEMENTATION

DESIGN OF PSEUDO-RANDOM SEQUENCE GENERATOR

1. Pseudo-Random Number Generators (PRNGs)

As the word 'pseudo' suggests, pseudo-random numbers are not random in the way you might expect, at least not if you're used to dice rolls or lottery tickets. Essentially, PRNGs are algorithms that use mathematical formulae or simply recalculated tables to produce Sequences of numbers that appear random. A good example of a PRNG is the linear congruential method. A good deal of research has gone into pseudo-random number theory, and modern algorithms for generating pseudo-random numbers are so good that the numbers look exactly like they were really random.

The basic difference between PRNGs and TRNGs is easy to understand if you compare computer-generated random numbers to rolls of a die. Because PRNGs generate random numbers by using mathematical formulae or precalculated lists, using one corresponds to Someone rolling a die many times and writing down the results. Whenever you ask for a die roll, you get the next on the list. Effectively, the numbers appear random, but they are really predetermined. TRNGs work by getting a computer to actually roll the die — or, more commonly, use some other physical phenomenon that is easier to connect to a computer than a die is.

PRNGs are *efficient*, meaning they can produce many numbers in a short time, and *deterministic*, meaning that a given sequence of numbers can be reproduced at a later date if the starting point in the sequence is known. Efficiency is a nice characteristic if your application needs many numbers, and determinism is handy if you need to replay the same sequence of numbers again at a later stage. PRNGs are typically also *periodic*, which means that the sequence will eventually repeat itself. While periodicity is hardly ever a desirable characteristic, modern PRNGs have a period that is so long that it can be ignored for most practical purposes.

These characteristics make PRNGs suitable for applications where many numbers are required and where it is useful that the same sequence can be replayed easily. Popular examples of such applications are simulation and modeling applications. PRNGs are not suitable for applications where it is important that the numbers are really unpredictable, such as data encryption and gambling.

It should be noted that even though good PRNG algorithms exist, they aren't always used, and it's easy to get nasty surprises. Take the example of the popular web programming language PHP. If you use PHP for GNU/Linux, chances are you will be perfectly happy with your random numbers. However, if you use PHP for Microsoft Windows, you will probably find that your random numbers aren't quite up to scratch as shown in this visual analysis from 2008. Another example dates back to 2002 when one researcher reported that the PRNG on MacOS was not good enough for scientific simulation of virus infections. The bottom line is that even if a PRNG will serve your application's needs, you still need to be careful about which one you use.

2. True Random Number Generators (TRNGs)

In comparison with PRNGs, TRNGs extract randomness from physical phenomena and introduce it into a computer. You can imagine this as a die connected to a computer, but typically people use a physical phenomenon that is easier to connect to a computer than a die is. The physical phenomenon can be very simple, like the little variations in somebody's mouse movements or in the amount of time between keystrokes. In practice, however, you have to be careful about which source you choose. For example, it can be tricky to use keystrokes in this fashion, because keystrokes are often buffered by the computer's operating system, meaning that several keystrokes are collected before they are sent to the program waiting for them. To a program waiting for the keystrokes, it will seem as though the keys were pressed almost simultaneously, and there may not

be a lot of randomness there after all. However, there are many other ways to get true randomness into your computer. A really good physical phenomenon to use is a radioactive source. The points in time at which a radioactive source decays are completely unpredictable, and they can quite easily be detected and fed into a computer, avoiding any buffering mechanisms in the operating system. The Hot Bits service at Fourmilab in Switzerland is an excellent example of a random number generator that uses this technique. Another suitable physical phenomenon is atmospheric noise, which is quite easy to pick up with a normal radio. This is the approach used by RANDOM.ORG. You could also use background noise from an office or laboratory, but you'll have to watch out for patterns. The fan from your computer might contribute to the background noise, and since the fan is a rotating device, chances are the noise it produces won't be as random as atmospheric noise.

IV. SECURE HASH ALGORITHM

Secure hash This Standard specifies five secure hash algorithms, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. All five of the algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a message digest. These algorithms enable the determination of a message's integrity: any change to the message will with a very high probability, result in a different message digests. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers or bits.

Each algorithm can be described in two stages: preprocessing and hash computation. Preprocessing involves padding a message, parsing the padded message into m-bit blocks, and setting initialization values to be used in the hash computation. The hash computation generates a message schedule from the padded message and uses that schedule, along with functions, constants, and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest.

The five algorithms differ most significantly in the security strengths that are provided for the data being hashed. The security strengths of these five hash functions and the system as a whole when each of them is used with other cryptographic algorithms, such as digital signature algorithms and keyed-hash message authentication codes, can be found in [SP 800-57] and [SP 800-107].

Additionally, the five algorithms differ in terms of the size of the blocks and words of data that are used during hashing. Figure 1 presents the basic properties of these hash algorithms.

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512

Figure 1: Secure Hash Algorithm Properties

SECURE HASH STANDARD:

Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology (NIST) after approval by the Secretary of Commerce pursuant to Section 5131 of the Information Technology Management Reform Act of 1996 (Public Law 104-106), and the Computer Security Act of 1987 (Public Law 100-235).

1.Name of Standard: Secure Hash Standard (SHS) (FIPS PUB 180-3).
2.Category of Standard: Computer Security Standard, Cryptography.
3. Explanation: This Standard specifies five secure hash algorithms - SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 - for computing a

condensed representation of electronic data (message). When a message of any length less than 264 bits (for SHA-1, SHA-224 and SHA-256) or less than 2128 bits (for SHA-384 and SHA-512) is input to a hash algorithm, the result is an output called a message digest. The message digests range in length from 160 to 512 bits, depending on the algorithm. Secure hash algorithms are typically used with other cryptographic algorithms, such as digital signature algorithms and keyed-hash message authentication codes, or in the generation of random numbers (bits).

The five hash algorithms specified in this Standard are called secure because, for a given algorithm, it is computationally infeasible 1) to find a message that corresponds to a given message digest, or 2) to find two different messages that produce the same message digest. Any change to a message wills, with a very high probability, result in a different message digests. This will result in a verification failure when the secure hash algorithm is used with a digital signature algorithm or a keyed-hash message authentication algorithm.

4. Approving Authority: Secretary of Commerce.

5. Maintenance Agency: U.S. Department of Commerce, National Institute of Standards and Technology (NIST), Information Technology Laboratory (ITL).

6. Applicability: Secure hash Standard is applicable to all Federal departments and agencies for the protection of sensitive unclassified information that is not subject to Title 10 United States Code Section 2315 (10 USC 2315) and that is not within a national security system as defined in Title 44 United States Code Section 3502(2) (44 USC 3502(2)). This standard shall be implemented whenever a secure hash algorithm is required for Federal applications, including use by other cryptographic algorithms and protocols. The adoption and use of this Standard is available to private and commercial organizations.

7. Specifications: Federal Information Processing Standard (FIPS) 180-3, Secure Hash Standard (SHS) (affixed).

8. Implementations: The secure hash algorithms specified herein may be implemented in software, firmware, hardware or any combination thereof. Only algorithm implementations that are validated by NIST will be considered as complying with this standard. Information about the validation program can be obtained at http://csrc.nist.gov/groups/STM/index.html.

9. Implementation Schedule: Guidance regarding the testing and validation to FIPS 180-3 and its relationship to FIPS 140-2 can be found in IG 1.10 of the Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program at ttp://csrc.nist.gov/groups/STM/cmvp/index.html.

10. Patents: Implementations of the secure hash algorithms in this standard may be covered by U.S. or foreign patents.

11. Export Control: Certain cryptographic devices and technical data regarding them are subject to Federal export controls. Exports of cryptographic modules implementing this standard and technical data regarding them must comply with these Federal regulations and be licensed by the Bureau of Export Administration of the U.S. Department of Commerce. Information about export regulations is available at: http://www.bis.doc.gov/index.htm.

12. Qualifications: While it is the intent of this Standard to specify general security requirements for generating a message digest, conformance to this Standard does not assure that a particular implementation is secure. The responsible authority in each agency or department shall assure that an overall implementation provides an acceptable level of security. This Standard will be reviewed every five years in order to assess its adequacy.

13. Waiver Procedure: The Federal Information Security Management Act (FISMA) does not allow for waivers to Federal Information Processing Standards (FIPS) that are made mandatory by the Secretary of Commerce.

14. Where to Obtain Copies of the Standard: This publication is available electronically by accessing http://csrc.nist.gov/publications/. Other computer security publications are available at the same web site.

Some application may require a hash function with a message digest length different than those provided by the hash functions in this Standard. In such cases, a truncated message digest may be used, whereby a hash function with a larger message digest length is applied to the data to be hashed, and the resulting message digest is truncated by selecting an appropriate number of the leftmost bits. For guidelines on choosing the length of the truncated message digest and information about its security implications for the cryptographic application that uses it, see SP 800-107.

V. CONCLUSION

We are done the proposed topology and traditional LFSRs have similar statistical properties, it exhibits a very high linear complexity (LC) and the sequences generated from the novel topology are much more difficult to be predicted. The proposed cipher is described and design criteria to choose the key of the system are also given. As an example, a 16-bit length generator was implemented on a Xilinx FPGA and experimentally verified using the most common statistical and randomness tests. Then it was also designed in standard cell AMS technology. Moreover, to test the generator's inviolability, a novel methodology, based on a multiperceptron neural network, was also developed and is also presented.

Secure Hash standard specifies algorithm that can be used to generate digests of messages. The digests are used to detect whether messages have been changed since the digests were generated.

REFERENCES

[1] Abdel-hafeez.S.,Sawalmeh.A. and Bataineh.S.,"High Performance AES Design using Pipelining Structure over GF(28)" IEEE Inter Conf.Signal Proc and Com.,vol.24-27, pp.716-719,Nov. 2007

[2] J.Yang, J.Ding, N.Li and Y.X.Guo,"FPGA-based design and implementation of reduced AES algorithm" IEEE Inter.Conf. Chal Envir Sci Com Engin(CESCE).,Vol.02, Issue.5-6, pp.67-70, Jun 2010.

[3] A.M.Deshpande, M.S.Deshpande and D.N.Kayatanavar,"FPGA Implementation of AES Encryption and Decryption"IEEE Inter.Conf.Cont,Auto,Com,and Ener., vol.01,issue04, pp.1-6,Jun.2009.

[4] Hiremath.S. and Suma.M.S.,"Advanced Encryption Standard Implemented on FPGA" IEEE Inter.Conf. Comp Elec Engin.(IECEE),vol.02,issue.28,pp.656-660,Dec.2009.

[5] AI-Wen Luo, Qing-Ming Yi, Min Shi. "Design and Implementation of Area-optimized AES on FPGA" ,IEEE Inter.conf.chal sci com engin.,978-1-61284-109-0/2011.

[6] Rizk.M.R.M. and Morsy, M., "Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA", IEEE Inter Conf. Desig Tes Wor.,vol.1,issue.16,pp.207-217, Dec. 2007.

[7] Liberatori.M.,Otero.F.,Bonadero.J.C. and Castineira.J. "AES-128 Cipher. High Speed, Low Cost FPGA Implementation", IEEE Conf. Southern

Programmable ogic(SPL),vol.04,issue.07,pp.195-198,Jun. 2007.

[8] Abdelhalim.M.B., Aslan.H.K. and Farouk.H. "A design for an FPGAbased implementation of Rijndael cipher",ITICT. Ena Techn N Kn Soc.(ETNKS), vol.5,issue.6,pp.897-912,Dec.2005.

[9] Fedaral Information Processing Standards publication 197 November 26,2001"ADVANCED ENCRYPTION STANDARD (AES)".

[10] "Architectures and VLSI Implementations of the AES-Proposal" Rijndael. N. Sklavos and O. Koufopavlou,.IEEE TRANSACTIONS ON COMPUTERS, VOL. 51, NO. 12, DECEMBER 2002.

[11] NIST, "Advanced Encryption Standard (AES)",NIST,FIPS-197,2001.

[12] http : // Advanced Encryption Standard-wikipedia , the free encyclopedia.html.

[13] P.Chodowiec and K.Gaj." Very compact FPGA implementation of the AES algorithm". In Proc.5th Int, workshop on Cryptographic Hardware and Embedded systems (CHES 2003), pages 319-333, cologne,Germany,Sept,8-10,2003.

[14] P.Canright." A Very Compact S-box for AES".In Proc.7th Int, workshop on Cryptographic Hardware and Embedded systems (CHES 2005), pages 441-455, Edinberg, UK , Aug.29-Sept,2005.

[15] K..Viswanath, Dr P.v.Rao, Veeresh Patil,"Design and Implementation of Area-Optimized 256-bit Advanced Encryption Standard for real time images on FPGA", ISSN:2319-1112/V1N2:134-140©IJAEEE.

[16] Satoh .A.Morioka.S,Takano.k. and Munetoh .S,"A Compact Rilndael Hardware Architecture with S-box Optimization"LNCs 2248, ASIA CRYPT 2001,pp.239-254.2001.256.

[17] Ahmed Rady, Ehab EL Sehely, A.M.EL Hennawy,"Design and Implementation of area optimized AES algorithm on reconfigurable FPGA",IEEE Inter.conf.Comp Elec Engin(IECEE), 978-1-4244-1847-3/07.2007.

[18] S.Sankar Ganesh,J.Jean Jenifer Nesam,"FPGA Based SCA Resistant AES S-BOX Design",International Journal of Scientific & Engineering Research,volume4,Issue 4,pp.1143-1149,April-2013.

Authors Profile:

1. **CH .S.RANADHEER,**

M.Tech In VLSI System Design Assistant Professor, Department Of ECE, TRINITY COLLEGE OF ENGINEERING AND TECHNOLOGY, KARIMNAGAR.

2. RAMESH JITTY,

M.Tech in Electronics And Communication Engineering, Assistant Professor, Department of ECE, TRINITY COLLEGE OF ENGINEERING AND TECHNOLOGY, KARIMNAGAR.