Securing AES Accelerator from Key Leaking Trojans on FPGA

Dr.Pydimarri Padmaja
Professor,ECE Dept.,Teegala Krishna Reddy engineering college ,Hyderabad, INDIA

G shirisha
Asst,Professor, ECE Dept.,Teegala Krishna Reddy engineering college ,Hyderabad, INDIA

Abstract- Securing AES Accelerator from Key-Leaking Trojans on FPGA.Reconfigurable hardware presents a useful platform for building systems with high performance and secured nature. A new method for protecting 128-bit Advanced Encryption Standard (AES 128-bit) accelerator on Field Programmable Gate Array (FPGA) for embedded systems and cloud server is proposed. One of the major issues faced by the AES accelerator is the security of the key stored inside the FPGA memory. Security for the key inside the accelerator is provided through a masking scheme. To work with the masked key, a modified key expansion that maintains the throughput through a properly designed multistage pipelining is proposed. The proposed method takes the advantage of reconfigurable computing for flexible and efficient hardware implementation and provides security against key-leaking Trojans. The efficiency of the masked AES implementation is found to be 28.5 Mbps, which is 17.87% higher than the existing best wok. The security of the proposed masked scheme is validated through correlation and hamming distance calculation techniques.

Keywords – AES, Trojans, FPGA, Pipelining, Accelerator, Masking, Security, Encryption Key

I. Introduction

Advanced Encryption Standard (AES) is the widely using secure algorithm for encryption to provide privacy of data. Acceptance of cloud computing in every field causes increase in encryption load in cloud servers. To accelerate applications running on server and to reduce processor load, Field Programmable gate Arrays (FPGAs) are integrated with the server hardware. Computation-intensive applications can be shifted to FPGAs for increasing speed and reducing power consumption. FPGAs are reconfigurable hardware units that can be customized for required applications. Hence, high parallelism can be achieved with lower frequency. Cloud benefits from FPGA in several aspects. First, it could customize the FPGAs for computation-intensive application. Second, FPGAs could run with lower frequency and hence the heat production in server can be reduced to a large amount (Hauck & Andre, 2010; Kilts, 2007; Phan 2004; Teubner & Woods 2013).

Encryption is used in cloud for the privacy of data at rest and data in motion. That means disk encryption of user's VM, transfer of user data in encrypted form, encrypted communication between different users, encryption as a service, and so on (Amazon Web Services, 2016; Bokefode, Bhise, Satarkar, & Modani 2016; Krutz & Vines, 2010; CLOUDLINK, 2014; Cloudsigma; Encryption at Rest in Google Cloud, 2016; HP Atalla Cloud Encryption, 2013; Protecting Data in Microsoft Azure, 2014; Rahmani, Sundararajan, Ali, & Zin, 2013). FPGA accelerator can be used to speed up the encryption process for large amount of data. Use of FPGA will increase encryption speed and reduce power consumption. To get finest performance, the design should have high speed and low area consumption. Figure 1 shows the scenario in which FPGAs are used in cloud server as accelerators. The intellectual properties (IPs) can be collected from a hardware maker or from trusted third parties. When the processor assigns a job to an FPGA, the bitstream for hardware design can be loaded from bitstream storage if available or from outside cloud through external network.

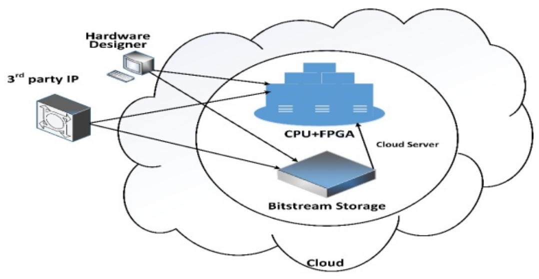


Figure 1. Usage of FPGA on cloud server

II. Proposed Algorithm

2.1 . PROPOSED KEY-MASKING TECHNIQUE

To tackle the key-leaking attack on AES accelerator, key-masking technique that prevents the leaking of secure key from memory is proposed. The method masks the key and subkeys values whenever they are stored in the memory inside FPGA. Further, when the algorithm needs it for securing or retrieving data for legitimate user, the proposed modified key expansion will work on the masked key during encryption and decryption.In Figure 2, the proposed masking key concept is shown. As soon as the secret key is given, the first Add Round Key step is allowed to use the key. Afterwards, masking is applied to the key before storing in FPGA memory. For further Add Round Key steps, the modified Expansion Key will produce the exact round key for it.
After a round key is produced, it is again masked in memory for producing next round key and this process continues.Masking is achieved by applying Substitute Byte operation on individual key blocks. The masked key can be securely stored and used in each AES round. The masking technique is shown in Equation (1).

(1)

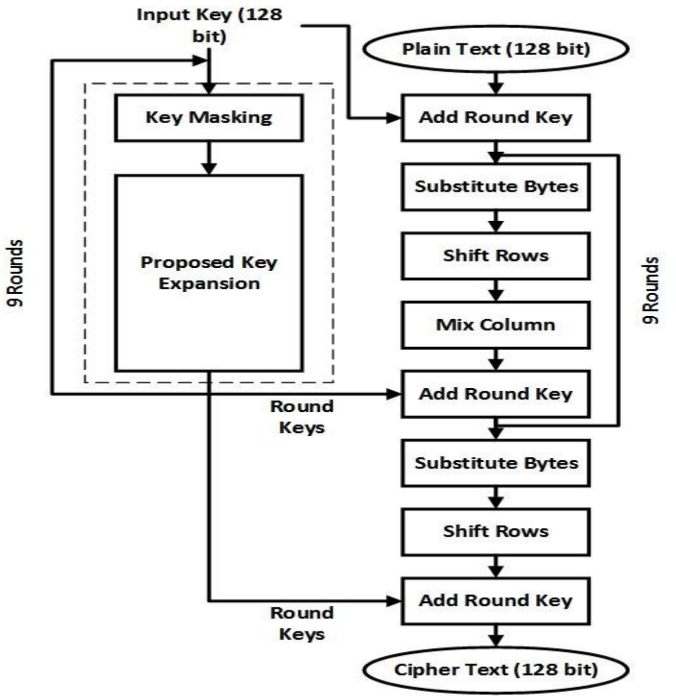
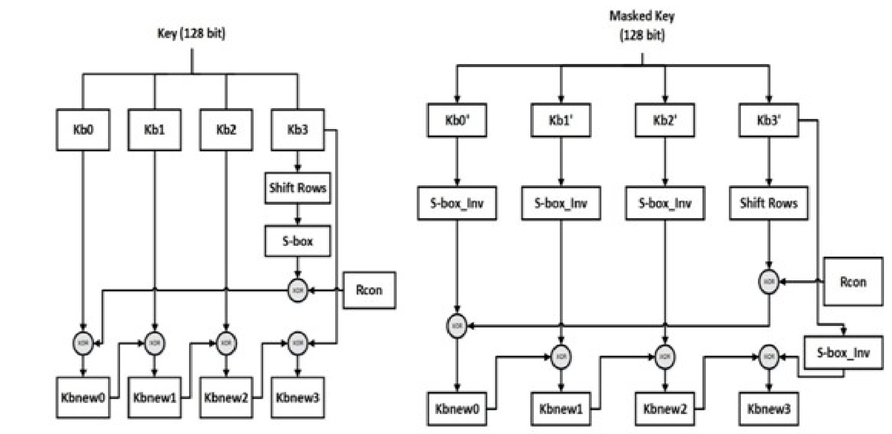


Figure 2. Proposed 128-bit Advanced Encryption algorithm with secure key masking

2.2. Proposed Key Expansion

AES Key Expansion takes 128-bit key, expands it to use in every Add Round Key step. It receives the key and divides it into four blocks—Kb0, Kb1, Kb2, and Kb3. Kb3 is taken, left circular shift by 1 bit, Substitute Byte is performed, XOR with constant Rcon (will be initially 01, then multiplied or divide by 02 at each round) and then XOR with Kb0 to get the new Kbnew0. Knew1, Kbnew2, and Kbnew3 are obtained by simply doing XOR Kb1, Kb2, and Kb3 with the previous output blocks (Lee et al., 2016). The process is shown in Figure 5(a). We propose a new Key Expansion technique that works on the masked stored key. The proposed Key Expansion works on the masked input key and gives the correct subkey to each Add Round Key in AES. The proposed technique provides security to stored input key and subkeys. After the subkey for a particular round is generated, it is again masked and stored for next-round key generation. The proposed Key Expansion is shown in Figure 3.The Key Expansion is done by taking the masked key and splitting into four blocks— Kb0′, Kb1′, Kb2′, and Kb3′. Then Kb3′ is taken, left circular shift, XOR with Rcon, and XOR with unmasked Kb0′ are applied to get new Kbnew0.



(a) Basic Key Expansion scheme (b) Proposed Key Expansion scheme

Figure.3.Key expansion

Here, Substitute Byte is not performed as it is already in substituted (masked) way. Kbnew1, Kbnew2, and Kbnew3 are obtained by performing XOR of the previous output with unmasked Kb1′, Kb2′, and Kb3′, as shown in Figure 10(b). It is given in Equation (2).

$$Kbnew0 = s\_box^{-1}(Kb0') \oplus [\mathrm{SR}(\mathrm{Kb3}') \oplus \mathrm{R}]$$

$$Kbnew1 = s\_box^{-1}(Kb1') \oplus \mathrm{Kbne}$$

$$Kbnew2 = s\_box^{-1}(Kb2') \oplus \mathrm{Kbne} \quad (2)$$

$$Kbnew3 = s\_box^{-1}(Kb3') \oplus \mathrm{Kbne}$$

Equations (3)–(6) show that the proposed Key Expansion

$$\begin{aligned} Kbnew0 &= s\_box^{-1}(Kb0') \oplus [\mathrm{SR}(\mathrm{Kb3}') \oplus \mathrm{Rcon}] \\ &= s\_box^{-1}(s\_box(Kb0)) \oplus [\mathrm{SR}(\mathrm{s\_box}(\mathrm{Kb3})) \oplus \mathrm{Rc} \\ &= Kb0 \oplus [\mathrm{SR}(\mathrm{s\_box}(\mathrm{Kb3})) \oplus \mathrm{Rc} \quad (3) \end{aligned}$$

$$\begin{aligned} Kbnew1 &= s\_box^{-1}(Kb1') \oplus \mathrm{Kbnew0} \\ &= s\_box^{-1}(s\_box(Kb1)) \oplus \mathrm{Kbne} \\ &= Kb1 \oplus \mathrm{Kbne} \quad (4) \end{aligned}$$

$$\begin{aligned} Kbnew2 &= s\_box^{-1}(Kb2') \oplus \mathrm{Kbnew1} \\ &= s\_box^{-1}(s\_box(Kb2)) \oplus \mathrm{Kbne} \\ &= Kb2 \oplus \mathrm{Kbne} \quad (5) \end{aligned}$$

$$\begin{aligned} Kbnew3 &= s\_box^{-1}(Kb3') \oplus \mathrm{Kbnew2} \\ &= s\_box^{-1}(s\_box(Kb3)) \oplus \mathrm{Kbne} \\ &= Kb3 \oplus \mathrm{Kbne} \quad (6) \end{aligned}$$

The memory where a masked key is stored is partitioned to allow parallel data access. The latency of the proposed Key Expansion after memory partitioning is found to be 4 and the initiation interval was found to be 5 for 1.23 ns path delay, meaning it takes four clock cycles to perform Key Expansion and the unit can be reused in the fifth clock cycle. Accordingly,a five-stage pipelining could make the initiation nterval to 1.

We are proposing a five-stage pipelining by dividing sub operations to fit into single clock cycle. Figure 4 shows the proposed five-stage pipelining for the proposed Key Expansion module.

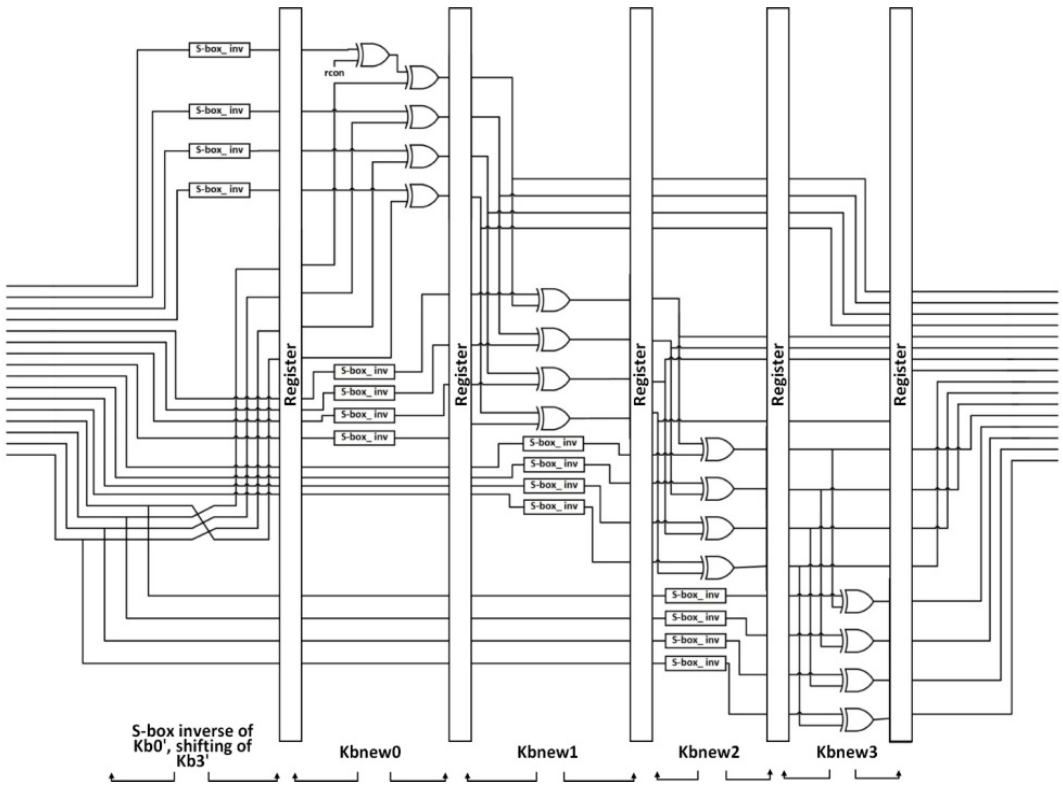


Figure .4. Proposed five-stage pipelined design of key expansion

III. IMPLEMENTATION AND RESULTS

The proposed method is tested on an XC7VX690T device. Vivado Design Suite—HLx Edition was used for the implementation (Vivado Design Suite, 2014 2013, 2013). Resource optimization was done by code optimization and careful placement of resources.Throughput, T can be calculated as given (Farashahi et al., 2014; Oukili & Bri, 2017):

Throughput and efficiency can be calculated from Equations (7)–(9). The number of bits processed by the system is 128 bits. Critical path delay for 813 MHz is 1.23 ns. The number of clock cycles that depend on throughput is 1 as at each clock cycle, the system could receive the input. So it does not depend on latency. Using the proposed scheme, a throughput of 104.06 Gbps was achieved. Table 3 gives the comparison of the proposed work and other related works.

The proposed system attained a throughput of 104.06 Gbps, which is same as that achieved by unmasked method reported in our previous work (Chellam & Natarajan, 2017) through proper pipelining. The work has been compared with all related FPGA-based AES designs to the extent of author’s knowledge. Table 3 shows the comparisons based on selected parameters and matrices. Compared to the previous works, it is clear that the efficiency achieved by the proposed design outperforms that of all related works. The 2-slow retiming technique by Farashahi et al. (2014) achieved a throughput of 82.47 Gbps with a maximum clock frequency of 671.524 MHz. Fault-tolerant design by Kamali and Hessabi (2016) produced a throughput of 86.5 Gbps, but taking more resources. The study by Oukili and Bri (2017) reported a throughput of 93.73 Gbps for unmasked design and 58.57 Gbps for masked design. As their masking scheme was to avoid power analysis attack and masking was applied to all functions, the resource utilization was much more.

Using memory partitioning and single-initiation interval-based multistage function pipelining, our technique attains a throughput of 104.06 Gbps at 813 MHz. For unmasked design, our work achieved an efficiency of 39.7 Mbps, which is higher than that achieved in all existing works. Our proposed masking scheme is aimed at avoiding key leakage through Trojans, and masking is applied only to key storage.

Table 3. Comparison of performance between related works and proposed work

		Platform	**No. of Bits**	**Frequency (MHz)**	**Path Delay (ns)**	**Throughput (Gbps)**	**Slices**	**Throughput/ Slice(Mbps/slice)**
Good and Benaissa (2007)		**XC3S4000-5**	**128**	**240.9**	**-**	**30835Mbps)**	**20720**	**1.4**
		XC2V8000-5	**128**	**222.8**	**-**	**28526 (Mbps)**	**31674**	**0.901**
Hammad et al. (2010)		**XC2V6000**	**128**	**305.1**	**-**	**39053 (Mbps)**	**10662**	**3.663**
Liu et al. (2013)		**XC7VX690T**	**128**	**516.8**	**-**	**66.1**	**3436**	**19.2**
Zhang and Parhi (2004)		**XCV1000**	**128**	**168.4**	**-**	**21.56**	**11022**	**1.95**
Granado-Criado et al. (2010)		**XC2V6000-6**	**128**	**194.7**	**5.136**	**24.922**	**3576**	**6.9**
Liu et al. (2015)		**XC7VX690T**	**128**	**593**	**1.6**	**75.92**	**4339**	**17.5**
Farashahi etal. (2014)	**4-level pipe**	**Virtex 5**	**128**	**433.06**	**-**	**55.432**	**3557**	**15.5**
	6-level pipe	**Virtex 5**	**128**	**528.37**	**-**	**67.631**	**3557**	**19.0**
Soltani and Sharifian (2015)	**two-slow retiming**	**Virtex 5**	**128**	**671.524**	**-**	**86**	**3557**	**24.1**
	coml-SpeM	**Virtex 5**	**128**	**764.059**	**1.3**	**97.8**	**10760**	**9.08**
	Roml-SpeM	**XC6VLX240T**	**128**	**764.059**	**1.3**	**97.8**	**10280**	**9.51**
	Com-Mux	**XC6VLX240T**	**128**	**803.988**	**1.2**	**102.91**	**28520**	**3.6**
Hussain and Jamal (2012)		**Virtex 7**	**128**	**456**	**-**	**5.3**	**2444**	**2.17**
Oukili and Bri (2017)	**Unmasked**	**XC6VLX240T**	**128**	**732.279**	**1.3**	**93.73**	**5759**	**16.2**
	Masked	**XC6VLX240T**	**128**	**457.582**	**2.1**	**58.57**	**9531**	**6.14**
Sharma et al. (2016)		**XC4VLX60-12(FF668)**	**128**	**476.19**	**-**	**59.52**	**3425**	**2.32**
Rahimunnisa et al. (2014)		**XC6VLX75T**	**128**	**505.5**	**-**	**37.1**	**1664**	**22.2**
Kamali and Hessabi (2016)		**XC7VX690T**	**128**	**675.62**	**-**	**86.5**	**4515**	**19.15**
Wang and Ha (2013)		**XC6VLX240T**	**128**	**319.29**	**-**	**40.9**	**9071**	**4.5**

IV.CONCLUSION

The use of reconfigurable hardware for high performance and secure application in computation-intensive environment is very useful. Usage of masking scheme for keys in AES accelerator provides protection against key-leaking Trojans, which is mandatory for security-critical applications. The modified Key Expansion technique was found to work properly with the masked key and gave same throughput with the proposed pipelining approach. The masked scheme could when pipelined, provides a throughput of 104.06 Gbps with an efficiency of 28.5 Mbps.
The security analysis shows that there is no correlation between the masked key and the original key.

REFERENCES

[1] Bokefode, J. D., Bhise, A. S., Satarkar, P. A., & Modani, D. G. (2016). Developing a secure cloud storage system for storing IoT data by applying role based encryption., *Procedia Computer Science 89*, 43–50.

[2] Bhasin, S., Danger, J. L., Guilley, S., Ngo, X. T., & Sauvage, L. (2013). Hardware Trojan horses in cryptographic IP cores. In *Workshop on Fault diagnosis and tolerance in cryptography (FDTC)* (pp. 15–29). IEEE.

[3] Chellam, M. B., & Natarajan, R. (2017). AES hardware accelerator on FPGA with improved throughput and resource efficiency. *Arabian Journal for Science and Engineering*, 1–18 Chellam, M. B., & Natarajan, R. (2017). AES hardware accelerator on FPGA with improved throughput and resource efficiency. *Arabian Journal for Science and Engineering*, 1–18

[4] Chiţu, C., & Glesner, M. (2005). An FPGA implementation of the AES-Rijndael in OCB/ECB modes of operation. *Microelectronics Journal*, *36*(2), 139–146.

[5] Farashahi, R. R., Rashidi, B., & Sayedi, S. M. (2014). FPGA based fast and high-throughput 2-slow retiming 128-bit AES encryption algorithm. *Microelectronics Journal*, *45*(8), 1014–1025.

[6] Freeman, J., & Young, T. (2009). Correlation coefficient: Association between two continuous variables. *Scope Tutorials,31–35*.

[7] Good, T., & Benaissa, M. (2007). Pipelined AES on FPGA with support for feedback modes (in a multi-channel environment). *IET Information Security*, *1*(1), 1–10.

[8] Granado-Criado, J. M., Vega-Rodríguez, M. A., Sánchez-Pérez, J. M., & Gómez-Pulido, J. A. (2010). A new methodology to implement the AES algorithm using partial and dynamic reconfiguration. *Integration 43*(1), 72–80.

[9] Hammad, I., El-Sankary, K., & El-Masry, E. (2010). High-speed AES encryptor with efficient merging techniques. *IEEE Embedded Systems Letters*, *2*(3), 67–71.

[10] [11] Hauck, S. & Andre, D. (2010). Reconfigurable computing: The theory and practice of FPGA-based computation (vol. 1) Morgan Kaufmann.

[11] Hennessy, J. L., & Patterson, D. A. (2011). *Computer architecture: A quantitative approach*. Elsevier.

[12] Heron, S. (2009). Advanced Encryption Standard (AES). *Network Security*, *2009*(12), 8–12.

[13] HP Atalla Cloud Encryption Securing Data in the Cloud (2013). Technical White Paper.