FOREX Publication
Open Access | Rapid and quality publishing

International Journal of
Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

Design of an Integrated Cryptographic SoC Architecture for Resource-Constrained Devices

Guard Kanda[1] and Kwangki Ryoo[2]

[1]*Department of Info, and Comm. Engineering, Hanbat National University, Daejeon, South Korea, guardkanda@gmail.com*
[2]*Department of Info, and Comm. Engineering, Hanbat National University, Daejeon, South Korea, kkryoo@gmail.com*

***Correspondence:** Kwangki Ryoo; kkryoo@gmail.com; Tel.: +82-10-5234-0569 (F.L.)

ABSTRACT- One of the active research areas in recent years that has seen researchers from numerous related fields converging and sharing ideas and developing feasible solutions is the area of hardware security. The hardware security discipline deals with the protection from vulnerabilities by way of physical devices such as hardware firewalls or hardware security modules rather than installed software programs. These hardware security modules use physical security measures, logical security controls, and strong encryption to protect sensitive data that is in transit, in use, or stored from unauthorized interferences. Without mechanisms to circumvent the ever-evolving attacking strategies on hardware devices and the data that they process or store, billions of dollars will always be lost to attackers who ply their trade by targeting such vulnerable devices. This paper, therefore, proposes an integrated cryptographic SoC architecture solution to this menace. The proposed architecture provides security by way of key exchange, management, and encryption. The proposed architecture is based on a True Random Number generator core that generates secret keys that are used in Elliptic Curve Diffie-Hellman Key Exchange to perform elliptic curve scalar multiplication to obtain public and shared keys after the exchange of the public keys. The proposed architecture further relies on a Key Derivation Function based on the CubeHash algorithm to obtain Derived Keys that provide the needed security using the ChaCha20_Poly1305 Authenticated Encryption with Associated (AEAD) Data Core. The proposed Integrated SoC architecture is interconnected by AMBA AHB-APB on-chip bus and the system is scheduled and controlled using the PicoRV32 opensource RISC-V processor. The proposed architecture is tested and verified on the Virtex-4 FPGA board using a custom-designed GUI desktop application.

General Terms: SoC, Cryptography, Hardware Security, RISC-V et. al.

Keywords: PicoRV32, ECC, FPGA, TRNG, AEAD_ChaCha20_Poly1305.

ARTICLE INFORMATION

Author(s): Guard Kanda and Kwanki Ryoo
Received: 11/04/2022; **Accepted**: 19/05/2022; **Published**: 10/06/2022;
e-ISSN: 2347-470X;
Paper Id: IJEER22TK406;
Citation: 10.37391/IJEER.100231
Webpage-link:
https://ijeer.forexjournal.co.in/archive/volume-10/ijeer-100231.html

Publisher's Note: FOREX Publication stays neutral with regard to Jurisdictional claims in Published maps and institutional affiliations.

1. INTRODUCTION

According to research by Statista in 2019, It is estimated that by 2025, up to about 75 billion devices will be connected to the internet [1]. This has been made possible due to IoT capability-extending technologies and platforms such as the 5G and the gigabit-fiber deployment. These IoT or ubiquitous devices have brought an enormous amount of flexibility and comfort to the lives of individuals. For instance, in a smart home, people can now sit in the comfort of their offices or place of work and turn on the air conditioning system in their homes to get the place well-conditioned before they arrive at home. Not all, smart homes can sense when their occupants are present and adjust the level of lightning in the home accordingly [29]. In the area of healthcare and fitness, connected IoT devices are constantly gathering data that opens a host of possibilities by way of an early and more accurate diagnosis of an ailment for better and effective treatment [2][30][31].

Although numerous benefits come with these IoT systems, devices, and platforms, certain key factors among many others are negatively influencing the growth and adoption of these systems. Three of these key factors are security, standards, and skill [3]. If the data being collected cannot be secured, then it becomes a challenge to adopt these devices. There are several implementation standards. Some are highly developed, others are conflicting and overlapping, and the rest are yet to be developed. Choosing the right standard for a particular purpose makes it a challenge. The final key inhibitor is Skill. The availability of adequately skilled and knowledgeable programmers. With all these sensors and their collected and processed data comes the highest need for securing these data which now translates to human lives. IoT security is becoming a major concern in recent years. The world has outgrown malware that steal private information. It now poses physical threats to its subscribers should these security barriers be breached.

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

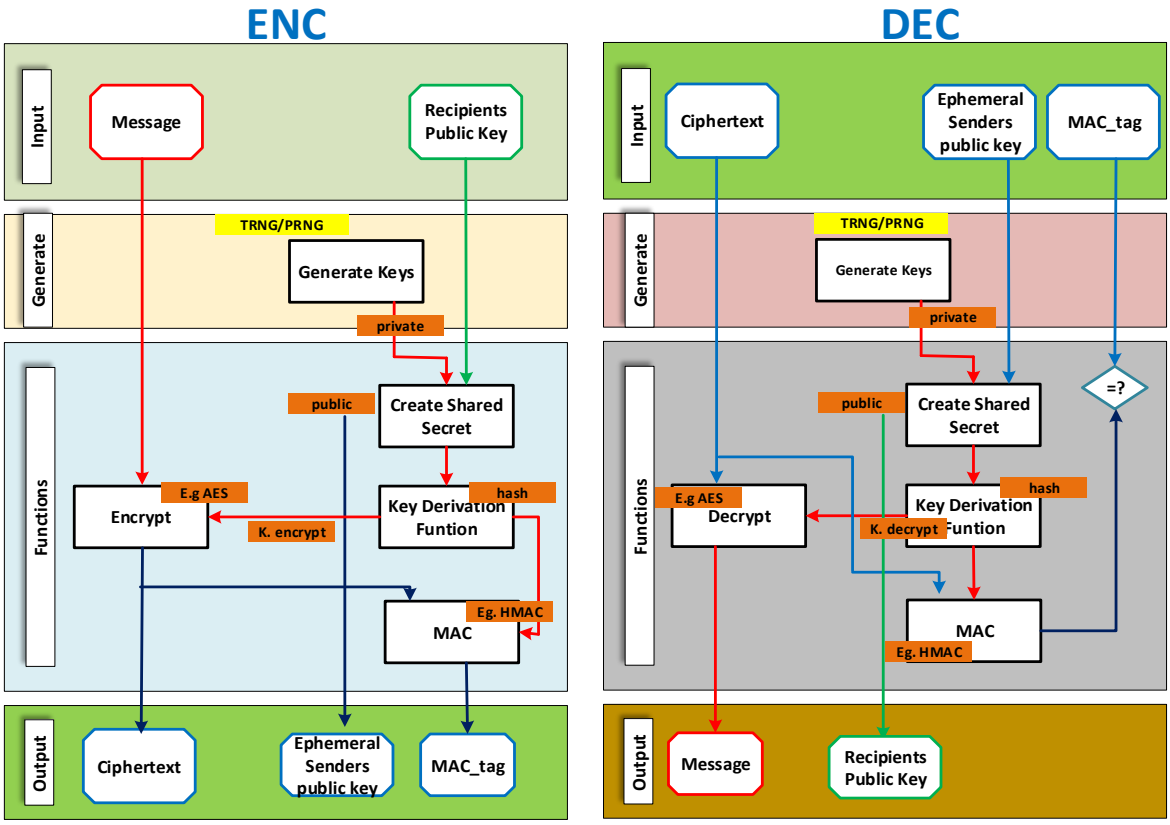

Figure 1: Overview of an Integrated Encryption Scheme

The greatest problem with these devices is that most of the manufacturers rolling out these devices have little to no measures in place to handle these device-related security risks and issues which increases at an alarming rate. Lack of security for these IoT devices can result in potentially catastrophic situations that can cost an individual, an organization, or a nation at large several fortunes.

Having such an exponential increase in the number of connected devices has led to a host of new and evolving highly sophisticated cybersecurity threats and attacks leading to information insecurity [4]. This has led to manners of such systems being on high alert and hardware security becoming one of the most critical parts of System-on-chip (SoC) design because of its usage for the internet of things (IoT) devices, cyber-physical systems, and embedded computing systems. This is the case because hackers will always try to find and exploit a hardware or software system that allows unrestricted access to assets or services mainly for financial gains. Since connected devices attacks are on the rise and are evolving by the day, it has left all interested parties vulnerable, that is both the consumer of these devices to the service providers and manufacturers. The ever-increasing complexity of on-chip components and long supply chain make SoCs vulnerable to hardware and software attacks. These attacks can be initiated either from inside the chip or from malicious software components.

Therefore, this paper presents an integrated cryptographic SoC architecture with operations similar to that of *Figure 1* and can provide an alternative solution to securing connected devices. In *Section 2*, a general discussion and brief overview of an integrated encryption scheme are presented. Details on the standardized Elliptic Curve Integrated Encryption Scheme (ECIES) are also presented. *Section 3* discusses the proposed integrated cryptographic System-on-a-Chip and its constituent IP cores. In *Section 4*, the simulation result of the proposed SoC together with its synthesis results are presented. The paper ends with the conclusion and future works in *Section 5*.

2. OVERVIEW OF ELLIPTIC CURVE INTEGRATED ENCRYPTION SCHEME

Discrete Logarithm Augmented Encryption Scheme (DLAES) [5] was initially presented in 1997 by Bellare Mihir and Rogaway Philip Diffie-Hellman. In the year 1998, it was jointly renamed the Augmented Encryption Scheme (DHAES) [6] by Michel Abdella and the authors in [5]. To avoid confusing the name in [6] with the Advanced Encryption Standard (AES), it was finally renamed Diffie-Hellman Integrated Encryption Scheme (DHIES) [7] in 2001, and, the integrated encryption scheme was proposed. The DHIES which was an extension of the ElGamal encryption protocol [8], integrated security primitives which included public and symmetric-key cryptographic algorithms and Message Authentication Code hash functions. DHIES was standardized in 2001 and was included in the ANSI X9.63 standard [9] with its subsequently modified versions in the 2004 IEEE 1363a standard [10].

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

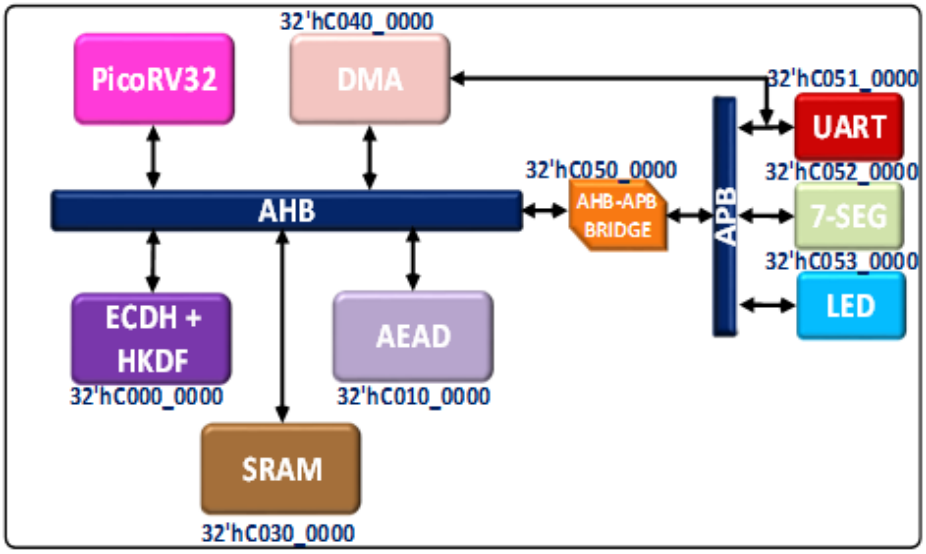

Figure 2: Proposed Cryptographic SoC Architecture

The ECIES is an amalgamation of encryption schemes [11] that interoperates to result in unified security. The integrated scheme comprises of the following functionalities:

1. Key Agreement Function such as ECDH is the function used to generate shared keys for communicating parties.
2. Key Derivation Function such as HKDF is used to generate a set of multiple keys for other functions in the encryption scheme.
3. Block Cipher such as AES, which is used for the actual encryption of data/information.
4. Hash Function such as SHA-1 is a function that always produces a fixed length of output data from any given input data.
5. Message Authentication Code is a code used for authentication by the various communicating parties

3. PROPOSED CRYPTOGRAPHIC SOC ARCHITECTURE

Figure 2 shows the proposed integrated cryptographic SoC architecture to equip hardware devices with secured means of communication and information exchange. The proposed architecture is based on a PicoRV32 RISC-V synthesizable processor as the embedded system processor that performs the system scheduling and control of the proposed integrated cryptographic SoC platform. The proposed architecture is an integration of previous works on ECC [12], TRNG [13], and AEAD_Chacha20_Poly1305 [14].

3.1 PicoRV32 Synthesizable Processor

PicoRV32 is an open-source hardware synthesizable CPU core that implements the RISC-V RV32IMC Instruction Set. PicoRV32 can be configured as RV32E, RV32I, RV32IC, RV32IM, or RV32IMC core, with an optionally built-in interrupt controller. PicoRV32 is core designed with optimization regarding the hardware area or size and the maximum operating frequency. For this reason, the PicoRV32 lacks any multi-stage pipelines and operates at maximum frequencies that range between 250-450 MHz based on test on the 7-Series of the FPGAs by Xilinx [15]. The input ports of the PicoRV32 shown in Figure 3 include the system clock—**clk**, active low reset—**resetn**, memory ready strobe signal—**mem_ready**, and a 32-bit wide **mem_rdata**, which is data read from the memory to the processor.

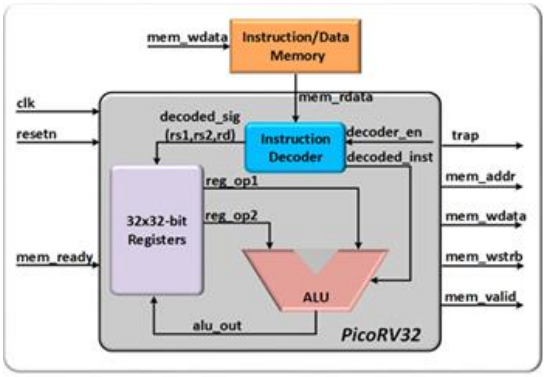

Figure 3: PicoRV32 RISC-V Simplified Block Architecture

PicoRV32's output ports include trap strobe signal which is asserted when the processor encounters an unfamiliar instruction, **mem_addr**, **mem_wdata**, a 3-bit byte strobe signal—**mem_wstrb**, and a valid data indication signal **mem_valid**. The PicoRV32I's area was further reduced by taking out the hardware multipliers and dividers that were designed as part of the original architecture since the proposed integrated SoC did not have use for them.

3.2 Elliptic Curve Diffie-Hellman (ECDH)

Two parties interested in exchanging communication based on a key agreement scheme are required to each provide some

FOREX Publication
Open Access | Rapid and quality publishing

International Journal of
Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

form of input or information to be used in creating a shared session key. The anonymous key agreement protocol allows two parties—popularly referred to as Alice and Bob—to both agree on an elliptic curve ***septuple (m, f(x), a, b, G, n, h)***. From the septuple and using algorithm from Table 1, the ECDH shared key [16]—which is a variant of the Diffie-Hellman key exchange [17]—can then be computed in 4 main stages. To generate the public key for each party involved in the communication requires the elliptic curve point multiplication computation discussed earlier. The following section introduces the proposed architecture of the ECSM processor core based on the doubling and additions of points discussed. The elliptic curve cryptography-based key possesses the advantage of shorter-length key size compared to other public-key cryptography such as RSA and the base of its security which is the discrete logarithm problem (DLP) which make the ECC a trapdoor function. Figure 4 shows us the various layers of abstraction involved in a single ECC protocol computation. At the very top of the layer is where we find the various implementation or usage of ECC. This layer, therefore, encompasses every layer beneath it. Beneath this layer is the scalar point multiplication layer which consists of the group operations of the point-double and point additions—the third layer from the top.

Table 1. ECDH Key Exchange Scheme

Steps	ECDH Action to Perform
1:	Receiver (Rx) and Transmitter (Tx) randomly generate random numbers between 1 and n (subgroup order—n) a_{Rx} and b_{Tx} (private keys) respectively
2:	Alice (Rx) and Bob (Tx) then generate individual public keys with the expression below H_{Rx} = (a_{Rx} x P) H_{Tx} = (b_{Tx} x P) where P is the base point (G) of the elliptic curve
3:	Alice (Rx) and Bob (Tx) can now exchange their public keys H_{Rx} and H_{Tx} over an unsecured channel
4:	Alice (Rx) and Bob (Tx) can now independently compute the agreed or shared key as follows $Sharedkey_{Rx}$ = a_{Rx} x H_{Tx} = a_{Rx} x (b_{Tx}x P) $Sharedkey_{Tx}$ = b_{Tx} x H_{Rx} = b_{Tx} x (a_{Rx}x P) i.e. $Sharedkey_{Tx}$ = $Sharedkey_{Rx}$

The point doubling and point additions operations also intern consist of the finite field arithmetic operation—this is the final layer in the stack—which performs the finite field addition, multiplication, and square and operations. The Elliptic Curve Scalar Multiplier (ECSM) architecture performs three key computations, these are transforming the coordinates from affine to projective domain, computing the doubling and additions, and then finally converting back to affine coordinate and extracting the resulting coordinates as shown in Figure 5. Based on the algorithm in Figure 5, the architecture in Figure 6 was proposed. As already stated, the key modular arithmetic modules that are utilized in this architecture are the finite field adder, multiplier, squarer, and divider. The finite field multiplication core is the most important module in the design of an ECC scalar point multiplication hardware architecture.

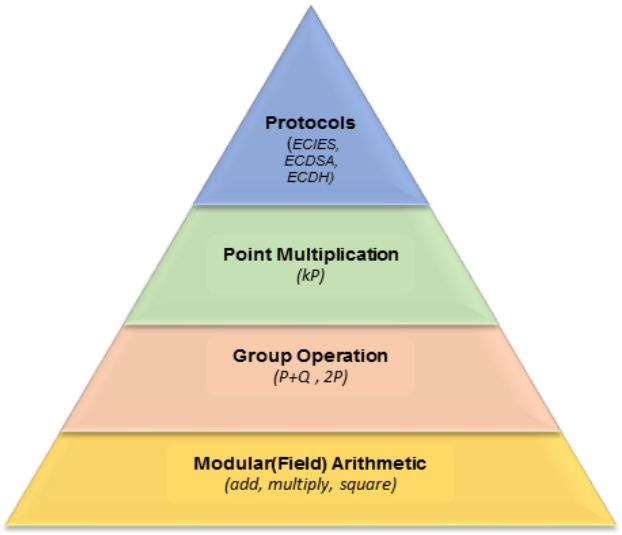

Figure 4: Abstraction Layers of ECC Protocols

Montgomery Ladder Scalar Multiplication Over $GF(2^m)$

Input: $P=(x,y), k=(1,k_{n-2},\ldots,k_1,k_0)$

Output: $Q=kP$

// initial stage : translate affine to projective domain

1 : $X_1 \leftarrow x_p, Z_1 \leftarrow 1, X_2 \leftarrow (x_p^4+b), Z_2 \leftarrow x_p^2$

//main Loop :Iteration for point doubling and addition

2 : for $j = n-2$ downto 0 do
3 : if $k_j = 1$, then
4 : $T \leftarrow Z_1, Z_1 \leftarrow (X_1 Z_2 + X_2 Z_1)^2, X_1 \leftarrow (x_p Z_1 + X_1 X_2 TZ_2)$;
$T \leftarrow X_2, X_2 \leftarrow (X_2^4 + bZ_2^4), Z_2 \leftarrow T^2 Z_2^2$;
5 : else
6 : $T \leftarrow Z_2, Z_2 \leftarrow (X_1 Z_2 + X_2 Z_1)^2, X_2 \leftarrow (x_p Z_2 + X_1 X_2 TZ_1)$;
$T \leftarrow X_1, X_1 \leftarrow (X_1^4 + bZ_1^4), Z_1 \leftarrow T^2 Z_1^2$
7 : endif
8 : end for

//Post-process to recover y-coordinate, translate projective to affine

9 : $x_3 \leftarrow X_1 / Z_1$
10 : $y_3 \leftarrow (x_p + x_3)\left[(X_1 + x_p Z_1)(X_2 + x_p Z_2) + (x_p^2 + y_p) Z_1 Z_2\right](x_p Z_1 Z_2)^{-1} + y_p$;
11 : return (x_3, y_3)

Figure 5: Montgomery Ladder Scalar Multiplication Over GF(2^m) Algorithm

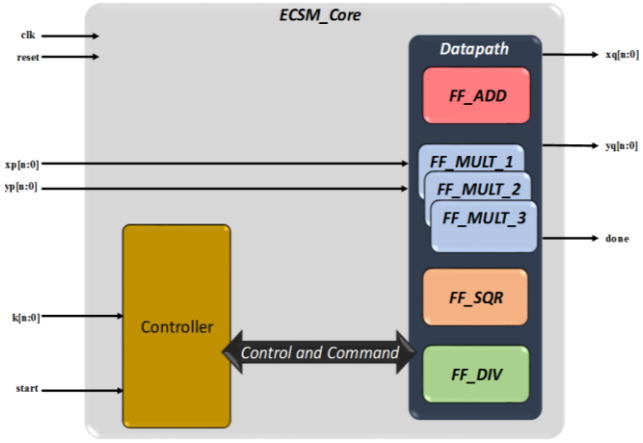

Figure 6: Montgomery Ladder Scalar Multiplication Over GF(2^m) Algorithm

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

The bit-serial approach implemented in this proposed architecture requires larger amounts of clock cycles to perform the field multiplication compared to the digit-serial approach. However, the area required for the digit-serial implementation is lesser than that of the combinational circuit but more than that of a bit-serial-based architecture. The ***ESCM*** architecture proposed was based on three (3) finite-field multipliers to improve the efficiency of the proposed architecture. As shown in the proposed scheduler for the controller, which is shown in *Figure 7*, the proposed ***ECSM*** core completes its point addition and doubling only in a time of $2M + 3$ clock cycles where M is the number of bits in the largest-sized operand.

Proposed schedule of the ECSM processor core for an M-bit operand
@cycle $1 \mapsto Z_A := (X_A + Z_A);\ m\mu lt1(X_A, Z_B);\ m\mu lt2(X_B, Z_A);\ m\mu lt3(X_A, Z_A)$
@cycle $2 \mapsto Z_B := Z_B^2$
@cycle $3 \mapsto Z_B := Z_B^2$
@cycle $M \mapsto T := m\mu lt1_{output};\ X_B := m\mu lt2_{output};\ X_A := m\mu lt3_{output}$
@cycle $M+1 \mapsto Z_A := (T + X_B);\ X_A := X_A^2$
@cycle $M+2 \mapsto Z_A := Z_A^2$
@cycle $M+3 \mapsto m\mu lt1(T, X_B);\ m\mu lt2(X_p, Z_A)$
@cycle $2M+2 \mapsto X_B := m\mu lt1_{output};\ T := m\mu lt2_{output}$
@cycle $2M+3 \mapsto X_A := Z_B;\ Z_A := X_A;\ X_B := (T + X_B);\ Z_B := Z_A$

Figure 7: Proposed ECSM Controller Schedule

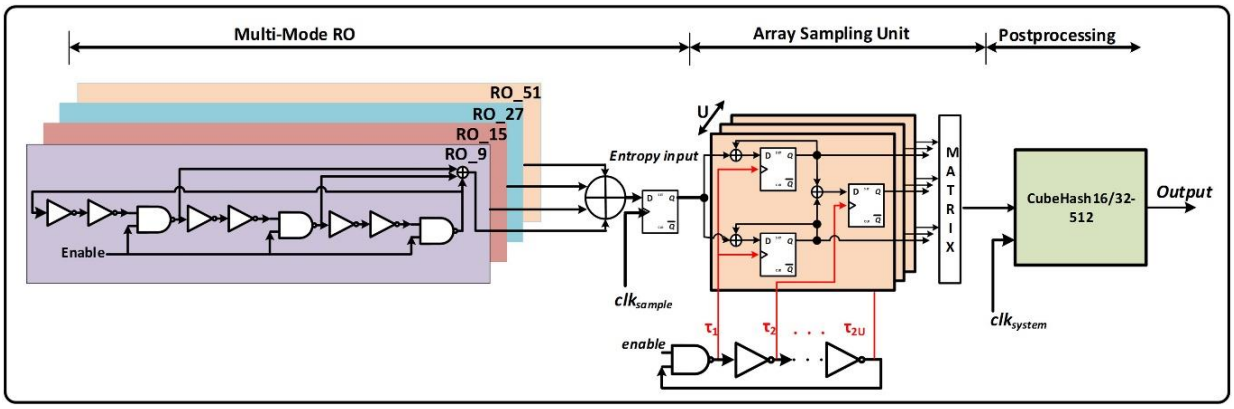

Figure 8: Proposed Multi-Edge Multi-Array Sampling TRNG Architecture

3.3 True Random Number Generator (TRNG)

As shown in the steps required to locally generate the same shared keys required for secured communication between parties, there is the need for a secret key generation in the first step of the algorithm in Table 1. In the proposed integrated cryptographic system. To achieve this, this paper proposes a true random number generator that uses basic and standard logic cells. The entropy source is based on a novel design of a multi-edge multi-mode ring-oscillator architecture shown in Figure 8. Each oscillator chain is made up of 3-multi-edge rings that are combined to form the oscillator chain. The use of the ring oscillators to build the architecture of the multi-edge entropy source as shown in Figure 8, increases the instability introduced into the bits sampled. The proposed architecture also includes a proposed multi-sampling unit that is simple to implement and is based on flip-flops. The proposed TRNG is cryptographically post-processed to obtain the final true random numbers. This paper opted for the cryptographic post-processing of bits because with this approach when the source of entropy ceases to function, the TRNG automatically becomes a pseudo-random number generator. A total of 1 GiB of data was generated based on the proposed TRNG architecture on a Spartan-6 FPGA equipped test board. The random samples were generated at 25MHz and 50MHz of sampling clock frequencies (Clk_{sample}) each. Since the proposed architecture does not embed an online test architecture, the sampled results were evaluated through NIST's statistical test suites[18] to establish if the generated bits possess the qualities that make them fit for use as TRNGs. The minimum pass rate for each statistical test except for the random excursion test was approximately 96 for the 100 binary sequences sample size. And from the results obtained success rates of above 0.96 were recorded. Not all, the ***P-values*** recorded are greater than 0.001, indicating that the bit sequences from the proposed TRNG passed using a significance level of alpha. In the ECDH module, the TRNG core generates either a 163-bit or 233-bit long secret key for the computation of the shared key process.

3.4 CubeHash-based HMAC KEY Derivation Function (HKDF) Core

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

The CubeHash is a collection of hash functions proposed and designed by Daniel J. Bernstein [19]. This set of hash functions was one of NIST's SHA-3 competition candidates that were eliminated in the second round although it is yet to be broken [20]. A key advantage of this algorithm is its simplicity. This hash algorithm uses a uniform structure for processing message digests of lengths of up to 512 bits, using a tweakable number of rounds and message block sizes. Six parameters namely parameters ***i, f, h, r, b***, and ***m*** specify the exact tweak or setup of the CubeHash algorithm. The ***i***-parameter specifies the number of rounds of the compression function to be executed to obtain the initialization vector. This parameter spans the range of 1 up to ∞ but is typically 16. The CubeHash notation is written as **CubeHash*i*+*r*/*b*+*f*-*h*(*m*)** to describe a specific variant of the algorithm. The parameter: ***i*** represents the number of rounds of compression to obtain the initialization vector, ***f*** denotes the number of round computations for the final message block, ***h*** denotes the width of the message digest which is typically between 8-bits to 512-bits. The parameter ***r*** represents the round compression for each message block, ***b*** determines the number of bytes per block of a message. Finally, ***m*** represents the length of the message that can be processed. The variant of CubeHash implemented in this research is the CubeHash160+16/64+32-512. *Figure 9* shows the top module of the implemented CubeHash message hashing function. The compression algorithm of the CubeHash shown in the drawn-out image to the right in *Figure 9* consists of 2 addition modulo-2^{32} operations, 2 XOR operations, 2 rotation operations, and 4 swapping operations. The round compression function—*Figure 9*—operates on the 1024-bit internal state, organized as 32 long words. Each of these 32 long words is 32-bits wide. The State is divided into two halves, each of size 512 bits and labeled as X and Y. This division is performed because the compression function only performs 10 simple operations on half of the internal state which is (512-bits) during each of the 10 compression rounds. At the end of each compression round the outputs X' and Y' are obtained from their respective X and Y halves. The X' and Y' outputs are fed back to X and Y if multiple rounds of the compression are required. Aside from being used as the cryptographic post-processing of the TRNG, the CubeHash algorithm was also employed in the derivation of keys that are used in the encryption of data or the generation of message authentication codes (MAC).

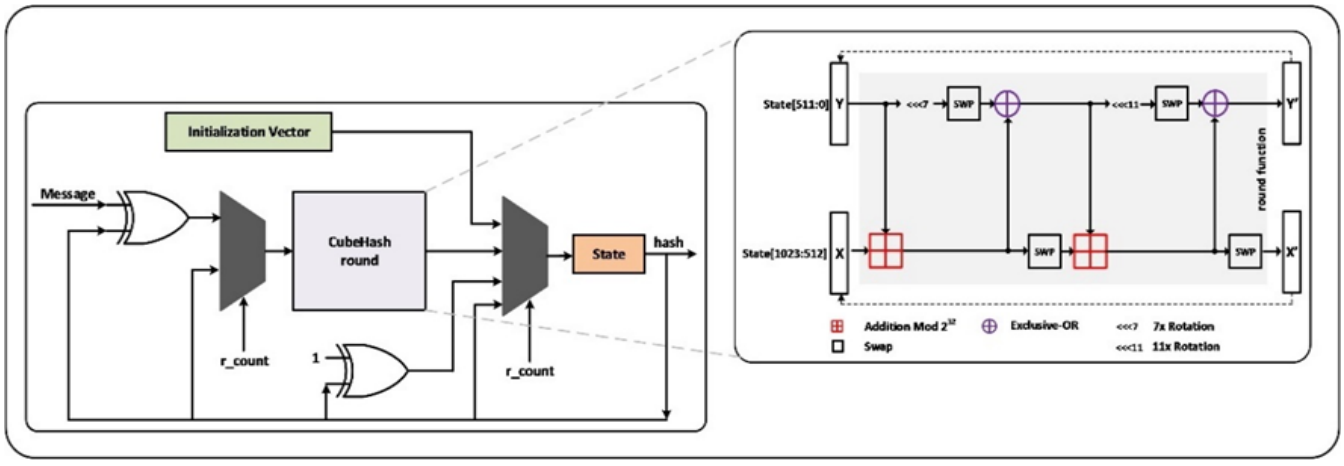

Figure 9: CubeHash Architecture for used in the Proposed TRNG and HMAC

The key derivation function employed is the HMAC-based Key Derivation Function (HKDF). This is a simple key derivation function that is based on the HMAC message authentication code. HKDF (RFC 5869) [21] follows the "extract-then-expand" phases. The first stage takes the keying material which is the shared key generated from ECDH and extracts from it a fixed-length pseudorandom key R. The second phase then expands the key R into several additional pseudorandom keys which become the output of the key derivation function. For the implemented HKDF in this paper, the hash algorithm that was used is the CubeHash shown in *Figure 9*. The value of info used for this HKDF is 163-bit ***0x7deedefefeefededeedefefeefededeedefefeefe***. The salt and input keying material (IKM) are the ECDH's generated y and x coordinates, respectively. As shown in *Figure 10*, the input key is padded to a 255-bit long key and used to extract the first key that is subsequently used in the "Expand Phase" of the HKDF. The final derived key is an addition of the previously generated output while the number of iterations is not realized.

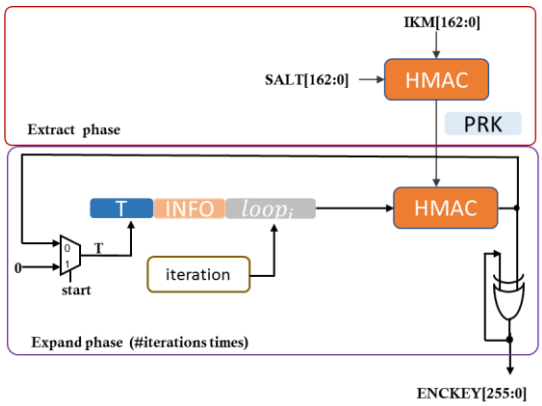

Figure 10: Proposed Implementation of HMAC-based Key Derivation Function

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

3.5 AEAD ChaCha20_Poly1305 Stream Cipher Architecture

The Cryptographic algorithms – ChaCha20 stream cipher [22] and Poly1305 [23] enhance security margins and achieve higher performance measures on a wide range of software platforms and have proven superior to its counterpart, the AES, in the software domain. This new stream cipher, compared to the benchmark AES, has recently been standardized but their implementations in hardware have had extraordinarily little to not very desirable results particularly in terms of area. In this paper, a compact, low-area, and high throughput ChaCha20-Poly1305 Authenticated Encryption with Associated Data (AEAD) architecture consisting of the ChaCha20 and Poly1305 algorithms are investigated and presented. The key area of improvement for the proposed hardware architecture is the simplified quarter-round design approach. This architecture uses the addition, rotation, and exclusive-or algorithms operators (gates).

Algorithm 1 : ChaCha20 Encryption

```
Inputs :
    256-bit encryption_key(K),
     32-bit block_counter (E),
     96-bit nonce(N),
      n-bit plaintext(M)
Output :
      n-bit ciphertext(C)

 1:  //Initialize Block
 2:  b ← [constant, K,E,N]
 3:  b' ← b
 4:  for i=1 to 10 do
 5:     quarterRound(b[0],b[4],b[8], b[12])
 6:     quarterRound(b[1],b[5],b[9], b[13])
 7:     quarterRound(b[2],b[6],b[10],b[14])
 8:     quarterRound(b[3],b[7],b[11],b[15])
 9:     quarterRound(b[0],b[5],b[10],b[15])
10:     quarterRound(b[1],b[6],b[11],b[12])
11:     quarterRound(b[2],b[7],b[8], b[13])
12:     quarterRound(b[3],b[4],b[9], b[14])
13:  endfor
14:  b ← b ⊞ b'
15:  C ← b ⊕ M
16:  return C
```

Figure 11: ChaCha20 Encryption Pseudo-Code Listing

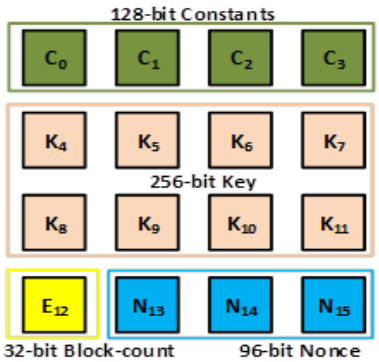

Figure 12: Initial State Matrix Setup for the ChaCha20 Stream Cipher

The ChaCha20 algorithm, shown in the listing of *Figure 11*, is composed of the main core round algorithm, known as the Quarter-Round operation. This algorithm works on a 4x4 matrix each of 32-bits shown in *Figure 12*, resulting in a total of 512-bit data. The upper-left of the matrix is marked index-0 and the bottom right is marked index-15. The ChaCha20—as can be deduced from the name—requires a total of 20 rounds to obtain the final keystream used to create the stream cipher. The rounds are executed as column and diagonal rounds alternatively. The upper 128-bits of the initial state matrix setup shown in *Figure 12* is filled with the constant of the ASCII converted sentence "expand 32-byte k." The next 256-bits which form the middle section of the initial state matrix contain the key for the encryption or decryption of data. This is followed by a 32-bit block counter. This block counter uniquely identifies every 64-byte (512-bit) block of data. With the 32-bit count value, a maximum of 256-gigabyte of data can be encrypted. The nonce which is the last 96-bit of the state matrix block is a unique number that is used to encrypt each block.

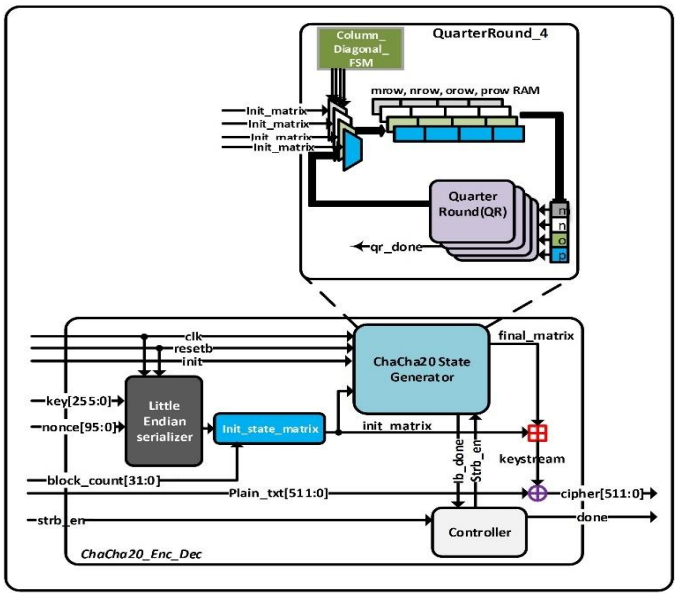

Figure 13: Proposed Hardware Architecture for ChaCha20 4xQR

Algorithm 2 : ChaCha20-Poly1305 Authenticator

```
Inputs : 256-bit keystream, message(msg)
Output : 128-bit Tag
 1:  //Poly1305
 2:  r       ←  keysteam [255:128]
 3:  r       ←  clamp(r)
 4:  s       ←  keysteam [127:0]
 5:  accm    ←  0
 6:  p       ←  (1<<130) - 5
 7:  for i=1 to msg_size/16 do
 8:    block ←  0x01 ⊔ msg[127:0]
 9:    accm  ←  accm + block
10:    accm  ←  (accm * r) mod p
11:  endfor
12:  Tag  ←  (accm + s) mod 2^128
13:  return Tag
```

Figure 14: Proposed Hardware Architecture for ChaCha20 4xQR

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

That is, the nonce should not be repeated for the same key. In other words, the nonce and the counter can be combined to perform the same purpose. This means that, effectively, a 128-bit nonce encrypts data of sizes above 256-gigabyte. To obtain a cipher using the ChaCha20 algorithm, the rounds are executed as column and diagonal rounds alternatively as shown in *Figure 11*. A total of twenty (20) rounds are required. The proposed architecture computes one diagonal and one column round in a cycle. In a pipeline fashion, the proposed architecture shown in *Figure 13* computes the ChaCha20 cipher in twenty (20) clock cycles rather than in eighty (80) The 256-bit input key and the nonce are passed through a little-endian serializer to convert the bits into little-endian form before being recombined into the initial state matrix. The initial state matrix then computes the final matrix which is also known as the keystream when the initial state matrix has been added after the twenty (20) clock cycles being controlled by the controller shown in *Figure 13*. At the end of the twenty (20) clock cycles, the plaintext is XORed with the keystream to obtain the stream cipher for the specified block of 64-bytes of data. If the data is larger than the 64-bytes, the count is increased by 1 to generate a unique set of the nonce that is used to identify each block of 64-bytes of data. The nonce is also increased changed for every key that is used to encrypt or decrypt data. Poly1305 module takes as input, 256-bit key, and an arbitrary-length message.

The 256-bit key to this module is partitioned into two halves as can be seen from the algorithm in *Figure 14*. The lower half of the key is assigned to the variable ***'r'*** and the upper half is assigned to the'*s*' variable. The value of ***'r'*** is clamped. The matrix for the clamping is a 4x4 matrix. Each vector location is an 8-bit value occupying 16 indexes to result in a total of 128 bits, the size of '*r*'. The upper left corner is indexed 15 and the lower right corner is indexed 0. The term OC represents Odd Clamp which is performed to clear the top four bits of that particular vector or matrix index to a value zero. The NC—No Clamp, represents the areas of the '*r*' vector that is not affected by the clamp. The final term which is the EC represents Even Clamp. The Even Clamp is performed to clear the bottom two bits of the value at that vector index or location. This clearing will make the value evenly divisible by 4. In the 4x4 matrix of **r**, ***r[3], r[7], r[11]*** and ***r[15]*** fall under the Odd Clamp, the ***r[4], r[8], r[12***] fall under the Even Clamp. To perform the clamp, A straightforward bitwise AND is performed on the vector *r* with the value 128-bit ***0x0ffffffc0ffffffc0ffffffc0fffffff***. The value of the prime number (P), used to perform the modulo computations, can be computed directly by performing a left shift of 130 on the value 1 and then subtracting 5 from the resulting value. This result is the 131-bit long hexadecimal number: ***0x3fffffffffffffffffffffffffffffffb***.

The Hardware Implementation of these algorithms focuses on improving these two core algorithms in terms of area, speed, and throughput. The ChaCha20 is employed to generate a keystream which is the result obtained after adding the initially constructed state matrix to the resulting matrix after the rounds of computation—this is 20 cycles for the 4xQR architecture or 80 cycles for the 1xQR architecture for this research. This keystream is then combined with the plaintext to obtain the ciphertext. At the core of ChaCha20's computation is what is known as the quarter-round computations. This structure can be implemented in several ways. Examination of the design in both pipeline and parallel architectures was performed. The design that used the pipeline approach reported a larger hardware area while improving operating frequency drastically. This is due to the reduction in the critical path of the architecture.

There are three main approaches to executing authenticated encryption. The form involves encrypting the data and then using portions of the encrypted data to generate a MAC tag known as the ***Encrypt-then-MAC***. The other form is generating a MAC from the plaintext or data to be encrypted. This MAC is then sent in addition to the encrypted plaintext (ciphertext) in what is known as the ***Encrypt-and-MAC***. The final is where the MAC is generated from the plaintext. The MAC and the plaintext are then combined to form new intermediate data. This intermediate data is then encrypted to form a ciphertext. This form is termed the ***MAC-then-Encrypt***. The ChaCha20-Poly1305 [24] implements a variant of this form of authenticated encryption used in TLS and its predecessor the SSL [25] [26]. The Associated Data that is appended to this form of authenticated encryption is to ensure it is contextually accurate. What this means is that moving a portion of a valid ciphertext to another portion will turn out to be invalid and cause its detection. The remaining architecture shown in *Figure 15*, was implemented using the two modules the ChaCha20 stream cipher and the Poly1305 authenticator is presented in this sub-section. The main components modules of the overall architecture use the individually built modules. Since this is a variant of ***MAC-then-Encrypt***, the key for the authentication is generated using the ChaCha20. For this key generation, the ***block_count*** is kept at zero. After the done signal is asserted, the keystream that is generated will be used to form the key to the Poly1305. The highest 256-bits of the keystream is captured and used as the poly1305 one-time key. The keys are clamped as explained in the section above and the Poly1305 module is enabled to begin execution. When the ***poly_done*** signal is asserted, we have a 128-bit value which will serve as our authentication tag for the specified batch of data being encrypted. The ***Main_Controller*** unit shown in *Figure 15* asserts the signal for the ChaCha20 module to be executed again to now encrypt the data. The same key, nonce but with the ***block_count*** now set to one and increases for each block. The increment can be linear or randomly generated this ensures that the effective nonce is different for each block of a 512-bit chunk of data to be encrypted. After this has been completed, the module ***AEAD_Recon_Data*** is enabled for a data reconstruction for the tag generation. The data is reconstructed by first placing the AAD data from bit zero upwards. This is followed by the 64-bit size of the AAD (AAD_size). Next in the concatenation is the ciphertext that has been generated and then finally the 64-bit little-endian integer representing the size of the ciphertext. The design can be parameterized to manage variable sizes. For this design, the message length used is 512-bit and the AAD utilized is 96-

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

bits. This implies that a reconstructed cipher data of size 736-bit long for the Poly_CSA. The total clock cycles required to generate the authentication tag and the ciphertext is 1350 cycles. This number of cycles is broken down as follows: 20 clock cycles required for generating the one-time-key for authentication, 20 clock cycles to generate the ciphertext, and about 1310 cycles required for the modulo reduction arithmetic.

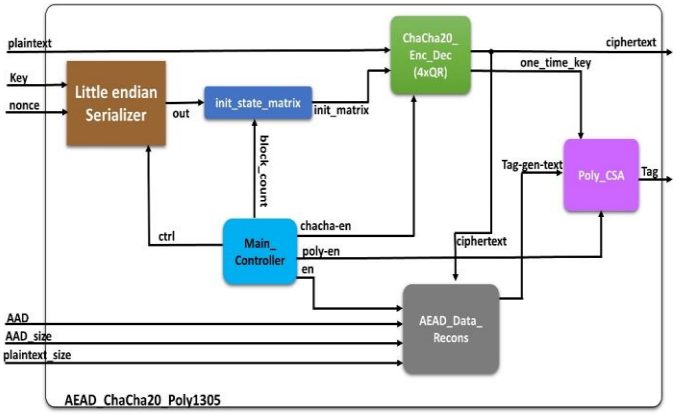

Figure 15: Block Diagram of the Proposed AEAD_ChaCha20_Poly1305 Architecture.

3.6 On-Chip Bus Communication Protocols

On-chip buses are not physical buses yet perform the function of interconnecting modules to enhance smooth communication and information interchange. Different SoC bus architectures exist. Some of these buses include the AMBA (Advanced Microcontroller Bus Architecture) from ARM. AMBA is a leading on-chip bus architecture that guarantees high performance for design. Bus arbitration techniques such as priority, round-robin, Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA). Three different bus architectures are defined under the AMBA specification. These are the Advanced High-Performance Bus (AHB), Advanced Systems Bus (ASB), and the Advanced Peripheral Bus (APB). AHB is suitable for high-performance designs, supports multiple bus-master operations, burst and split transfers, and wide data and address configuration. The APB is a peripheral bus used to connect low-speed peripherals. Between these two is the ASB. This is a cost-effective bus that allows multiple bus master operations and burst and pipelined transactions. The most current bus architecture is the Advanced eXtensible Interface (AXI) also from the ARM. The AXI is specifically intently designed for high-speed, high-performance, and high-frequency SoC designs. Notable features are the separate data and address phases, support for the unaligned transfer of data, and burst transfers. Multiple outstanding addressing and out-of-order transactions are equally supported. Arbitration schemes supported are the same as those supported by AMBA. In this paper, the bus communication protocol designed is the AHB bus for the system modules and then the APB for the peripheral communication with the UART, 7-Segment, and LEDs. The bus designed for the proposed SoC architecture has a data width of 32-bit and does not support all the modes of data transfers.

3.7 Additional SoC Architecture Components

The Direct Memory Access (DMA) controller is a simple module designed to perform the functionality of data read-write without the involvement of the system processor. The designed DMA accesses the on-chip bus to read and write data to the SRAM core. The DMA core has both master and slave bus interfaces. It operates as a bus master after the system processor hands over read-write functionalities to it through its slave interface. The DMA controller modeled in this paper has a direct connection to the UART core. This allows the DMA to transfer large data files between the SRAM and UART cores. The bus master hands over control of read-write data to the DMA by sending information regarding the start address of the transfer, the total amount of data to be transferred, and the transaction type which includes reading of plaintext from the UART to the SRAM, sending of plaintext or raw data to the AEAD core and writing the ciphertext back to SRAM and finally reading the ciphertext from the SRAM and sending it to UART. The UART core utilized in this SoC is the UART16550D [27] which is compatible with the industry-standard National Semiconductors' 16550A device. The UART core operates in either 8-bit or 32-bit data bus modes and operates in the FIFO-only mode. The UART core is also equipped with register level and functionality which is compatible with the NS16550A allowing the possibility of baud rate programing and FIFO size programming. The 7-Segment array is also added to the SoC architecture to display the bottom 32-bits of the Message Authentication Code Tag that is generated from the Poly1305.

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

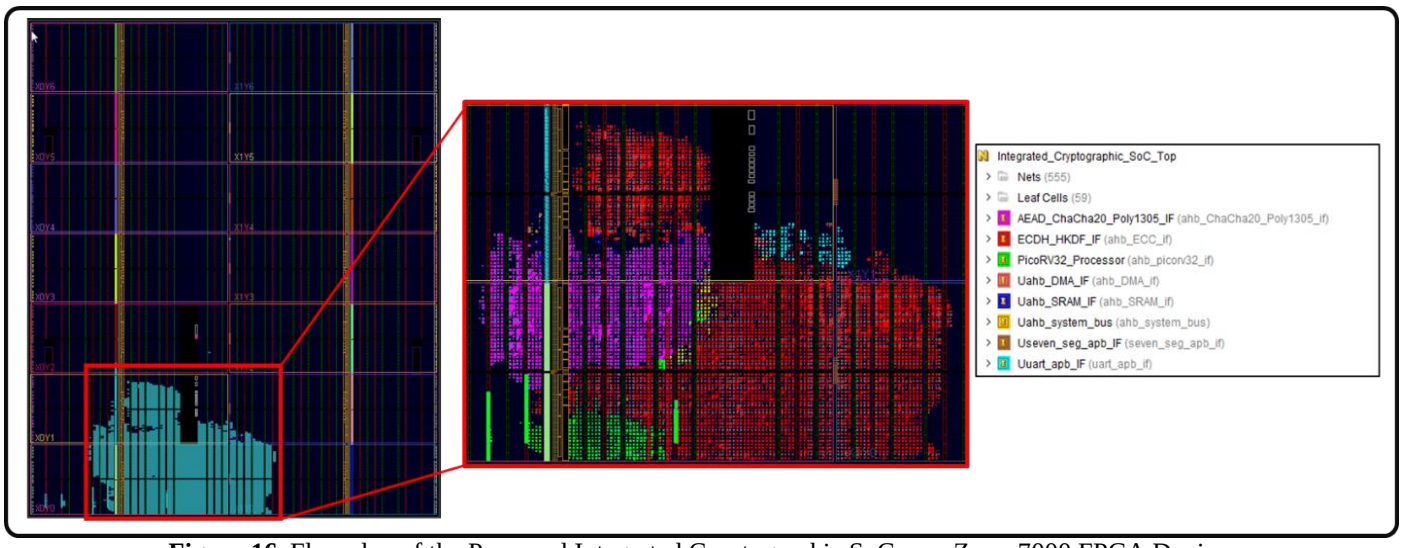

Figure 16: Floorplan of the Proposed Integrated Cryptographic SoC on a Zync-7000 FPGA Device.

4. HARDWARE SYNTHESIS RESULTS AND ANALYSIS

The elliptic curve-based integrated cryptographic SoC was designed using Verilog HDL. The proposed cryptographic SoC was synthesized using Precision RTL Synthesis Tool from Mentor Graphics. The proposed SoC architecture and its sub-IPs were simulated for both functional and timing correctness using the ModelSim 64-bit 10.6d standard edition. Synthesis results of the proposed integrated cryptographic SoC architecture are summarized in Table 2. The results are compared to a similar cryptographic core designed in [28]. Comparing the hardware resources required for similar modules in both implementations, it was observed that the proposed integrated cryptographic SoC, synthesized on similar Vertix-5 FPGA occupied about 80% fewer Slices compared to the TLS designed in [28]. Table 4 summarizes the hardware resources required by [28] and Table 3 summarizes the hardware resources required by similar algorithms or modules in the proposed SoC architecture. Except for the HMAC, the common modules in both designs occupied fewer resources in the proposed integrated cryptographic SoC architecture. The HMAC for the proposed integrated cryptographic SoC required more hardware resources because it was based on the CubeHash hashing function rather than the SHA-256 used by [28]. Additionally, the proposed SoC utilized no DSP blocks because all elliptic curve computations were not based on generic multiplies that were used in [28]. The results of the proposed cryptographic SoC architecture on the FPGA it was implemented and evaluated are shown in Table 2. It was observed that the proposed integrated cryptographic SoC architecture occupied 36.22% of the available 6822 Slices, 78.7% of the 27288 available LUTs, 36.21% of the 54576 total available Slice registers, and 28.42% of the 116 total available BRAM blocks. The design was implemented using 57 IO pins and 1 global buffer. Figure 16 shows the proposed SoC architecture's implementation on the Zynq-7000 xc7vx485tffg1153-3 device alongside the colour scheme showing the area resources utilized by the individual IP cores and how much area the proposed integrated core occupies relative to the FPGA resources available to the device. Additionally, Figure 17 shows the hierarchical view of the proposed cryptographic SoC architecture, illustrating the sub-cores or logics that make up the main core. Figure 17's colour scheme is based on the same colour scheme as that in Figure 16.

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of
Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

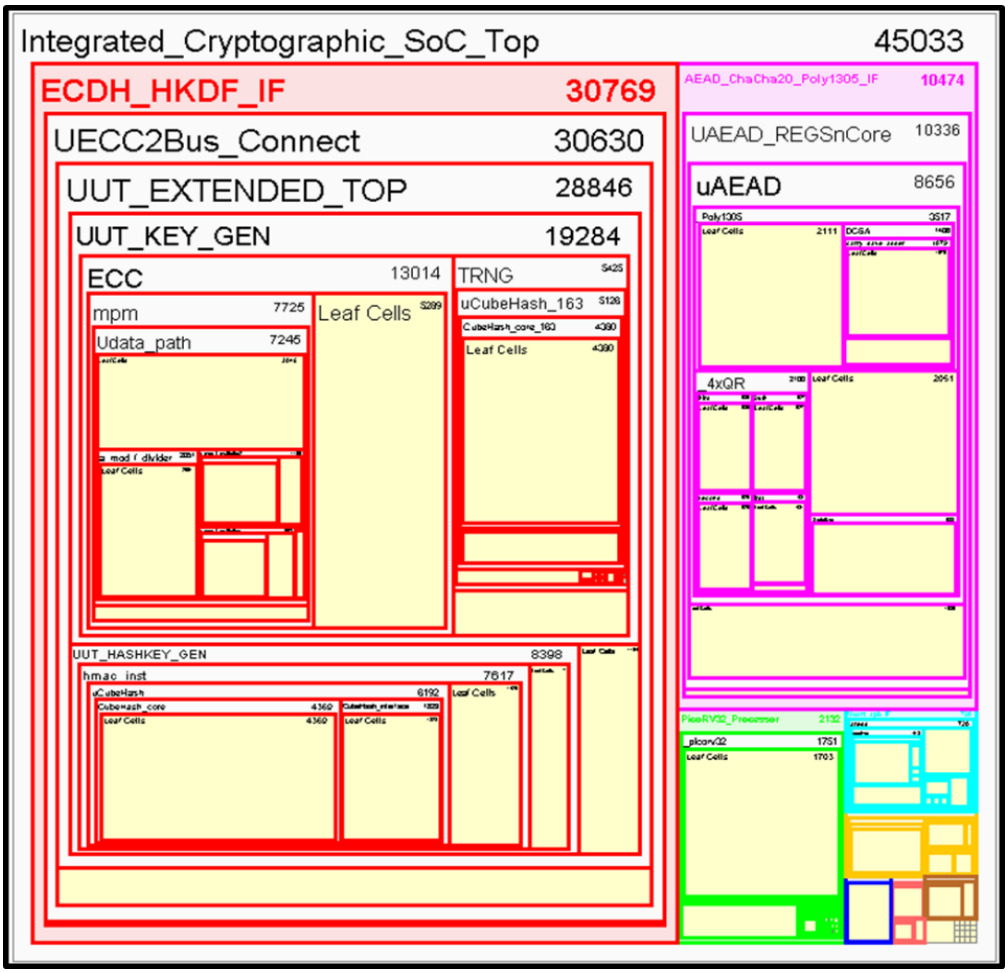

Figure 17: Hierarchical View of the Proposed Integrated Cryptographic SoC

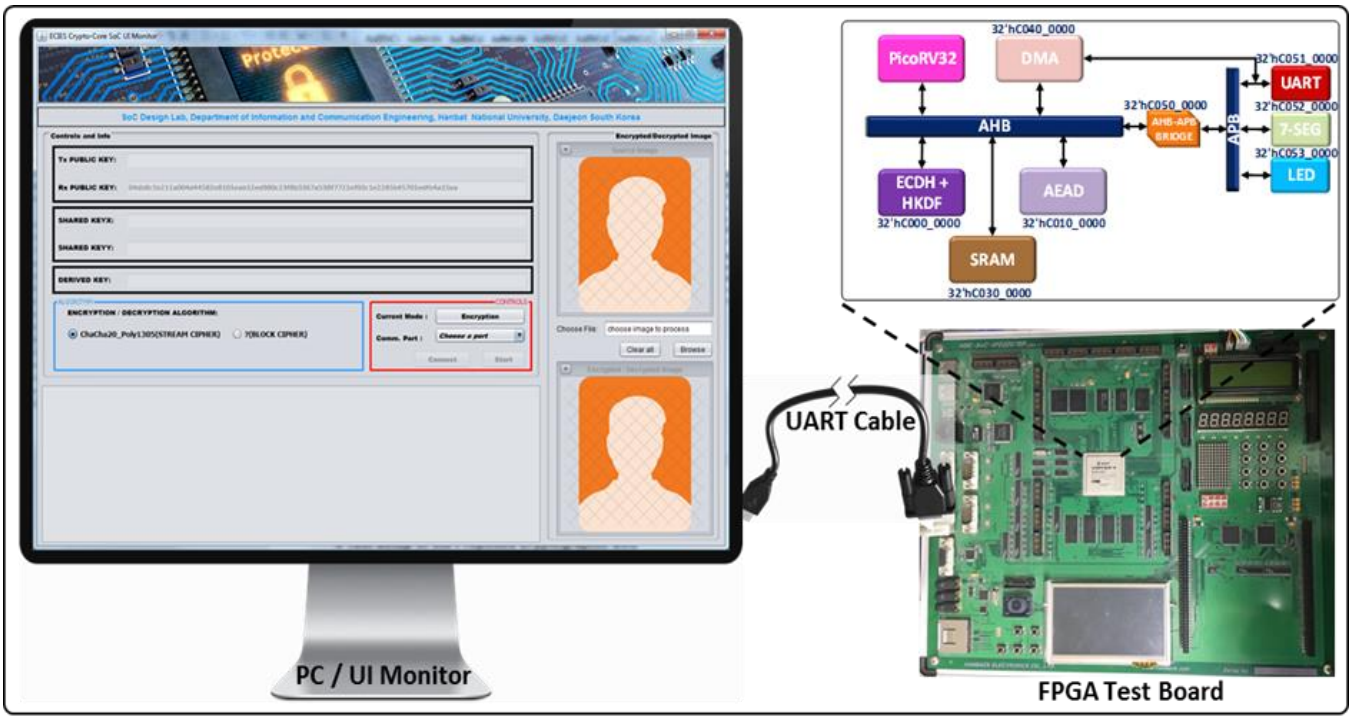

Figure 18: Proposed Integrated Cryptographic SoC FPGA-GUI Test Setup

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

Table 2. Summary of Synthesis of Proposed Integrated Cryptographic SoC on Virtex-4 FPGA Test Board

Proposed Designs	Area (Slices)	LUTs	Regs.	BRAM	DSP48E
PicoRV32I	843	1686	888	30	0
DMA Controller	310	619	542	0	0
ECDH + HKDF (ECC, TRNG, HMAC)	9370	18740	13137	1	0
SRAM_Controller	56	72	112	0	0
AMBA_AHB_APB	220	493	166	0	0
Seven-Segment Controller	31	54	62	0	0
UART	319	638	272	0	0
ChaCha20_Poly1305	540	4585	4320	0	0
Integrated Cryptographic SoC Top	13649	27298	19158	31	0

Table 3. Summary of Virtex-5 Synthesis Result for Similar Modules of TLS [28] Utilized in the Proposed Integrated Cryptographic SoC Architecture

Design	Area (Slices)	LUTs	Regs.	BRAM	DSP48E
TRNG	658	2301	2629	1	0
HMAC	267	852	1068	0	0
CubeHash	518	2168	2070	0	0
ECC	1171	7096	4684	0	0
Proposed Integrated Cryptographic SoC	4988	22240	19949	33	0

Table 4. Summary of Virtex-5 Synthesis Result of TLS Coprocessor [28]

Design	Area (Slices)	LUTs	Regs.	BRAM	DSP48E
TRNG	1170	3217	1700	0	1
HMAC	73	20	267	0	0
CubeHash	638	2325	1035	0	0
ECC	4624	12117	8234	13	0
Full TLS system with µP	14145	39052	29014	75	5

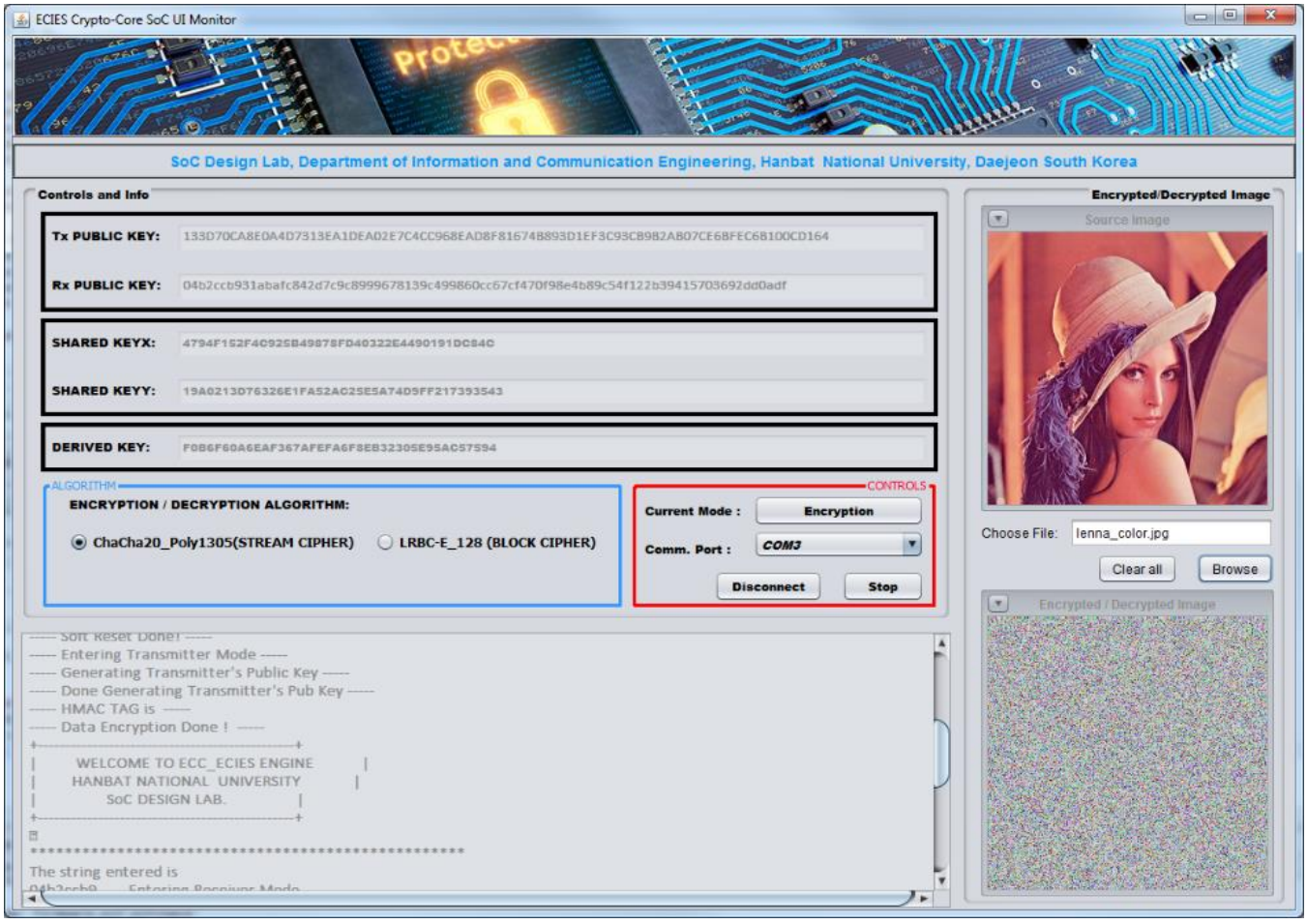

Figure 19: Test GUI Showing Key Generation and Encryption of Image

FOREX Publication
Open Access | Rapid and quality publishing
International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

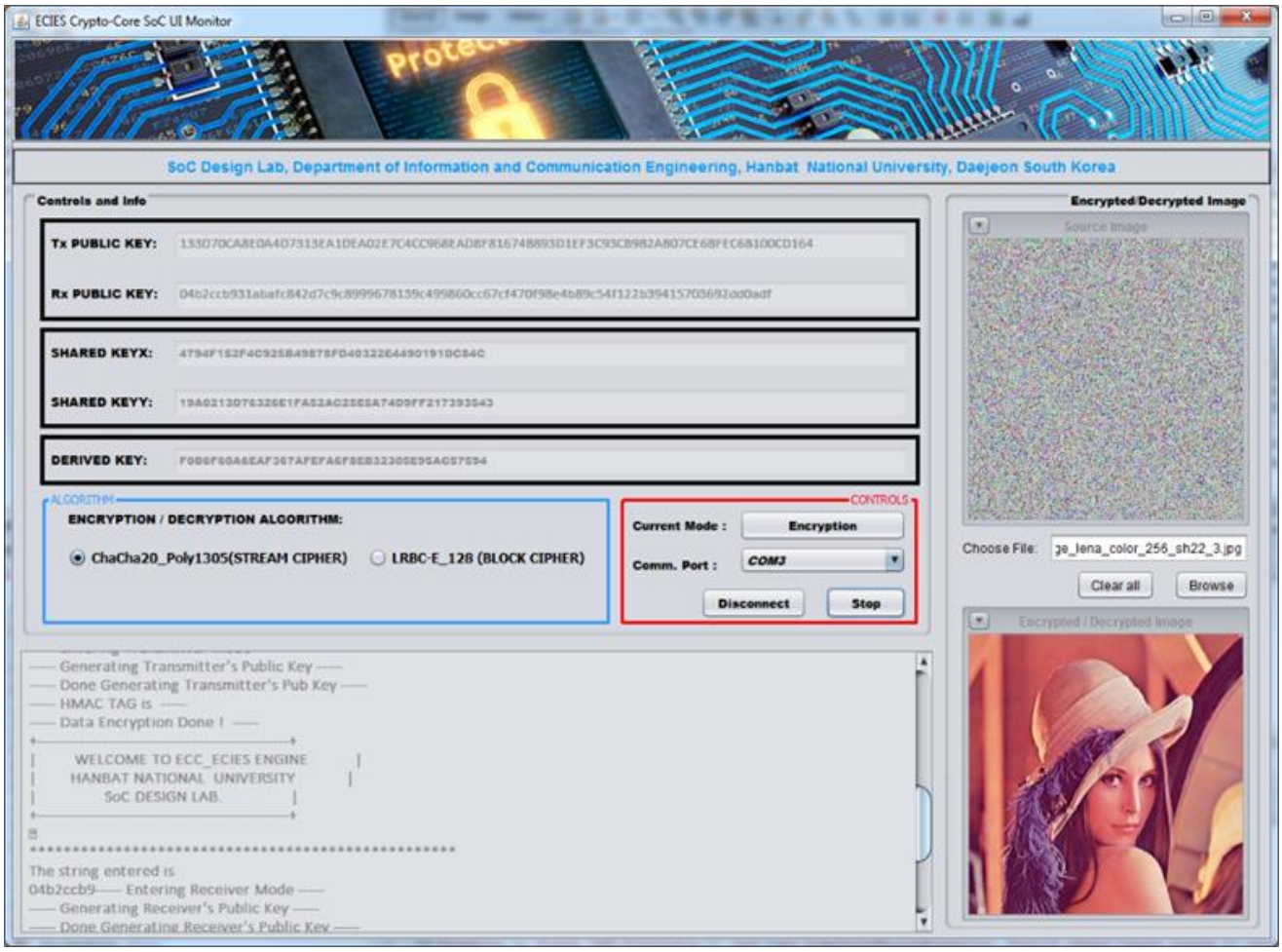

Figure 20: Test GUI Showing Successful Decryption of Encrypted Image with Same Keys

5. THE PROPOSED CRYPTOGRAPHIC SOC'S EXPERIMENTAL SETUP AND TEST

The FPGA test board—HBE-SoC-IPD— used in this research is equipped with the Virtex-4 FPGA device and was utilized in the test and verification process of the proposed. Cryptographic SoC design alongside all the available peripheral components that aided in the testing of the proposed integrated cryptographic SoC architecture. The setup for evaluating the proposed Integrated Cryptographic SoC is shown in Figure 18. The setup consists of the FPGA test board, a PC for running the test GUI to monitor the internal workings of the SoC, and finally a UART cable that interconnects the test GUI program and the FPGA test board. After the setup is completed, the user selects from the list of comm ports on the test GUI, the appropriate port on which the GUI communicates with the proposed cryptographic SoC. After all these selections, an image of size 128-by-128 pixels is selected and is displayed in the upper right corner of the Test GUI application. If the image does not fit the specified size—128x128, an error message is shown. At this stage, the connect button is clicked to establish a UART connection with the proposed integrated cryptographic SoC core. Upon successful connection, the text on the connect button changes to "disconnect" and the start button is enabled otherwise the text remains "connect" on the connect button and the start button remains disabled [32]. The start button is clicked to initialize data transfer. Since this is only a one-half test, the FPGA is regarded as the sender and the test GUI as the receiver. Hence, a randomly generated public key of the receiver is always generated when the test GUI is executed and is transferred to the processor. Once the processor gets the public key into a buffer, it sends this public key to the ***ECDH + HKDF*** module. The ***PicoRV32*** then programs the appropriate registers are to start the operation of the ***ECDH***. This first generates a TRNG—Secret Key. This secret key is then sent to the ***ECSM*** to compute the sender's public key—***Tx PUBLIC KEY***. Next, the receiver's public key—***Rx PUBLIC KEY*** and the sender's secret key are passed to the ***ECSM*** to compute the Shared Key. The shared key pair—***SHARED KEYX*** and ***SHARED KEYY***—are passed through the CubeHash based ***HKDF*** to generate the derived key—***DERIVED KEY—***that is used for the encryption of data. Whiles computing all of these keys, the selected image's byte data are extracted and transferred through the DMA block, to be stored to SRAM on the test board via the UART. Upon completion of the derived key computation, the sender's

International Journal of Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

public key, the shared key pair, and the derived key are all respectively updated on the Test GUI application through the PicoRV32 processor. This is done only for visualization of the keys during testing and may not be the case for a real-world implementation. The processor then sends the derived key based on the selected algorithm to the appropriate core for encryption/decryption. The processor then hands over control to the DMA where the image byte streams are read from the SRAM to the appropriate core and back to the SRAM after the processing is done 512-bits at a time for the AEAD_ChaCha20_Poly1305 but all in 32-bit chunks since the word size of the system is 32-bit. Figure 19 shows the successful reconstruction of the ***.jpg*** encrypted image data that is written back to the GUI test application and displayed in the lower right corner of the image. Keeping the same set of generated key parameters, the encrypted image's byte data are sent to the same selected cryptographic algorithm—AEAD_ChaCha20_Poly1305—and its successful decryption is shown in the bottom right corner of Figure 20. This validates the successful encryption and decryption crypto functionalities of the proposed integrated cryptographic SoC. It can be noted that the mode of operation in Figure 20 is *"Encryption"* as seen in Figure 19. This is because the AEAD_ChaCha20_Poly1305 does not require an encryption or decryption mode as the block cipher does. With the same encryption key, nonce, and initialization vector, an input plaintext will always result in its cipher and vice versa.

6. CONCLUSION AND FUTURE WORK

This paper proposed an integrated cryptographic encryption SoC architecture. The proposed SoC architecture integrated a RISC-V-based PicoRV32 synthesizable processor, a binary field domain elliptic curve Diffie-Hellman Key exchange and management algorithm, AEAD_ChaCha20_Poly1305, DMA controller, SRAM controller, and hardware peripherals including UART and 7-segment LED into a single core that interoperates seamlessly to provide security for IoT and ubiquitous devices that are resource-constrained. The proposed SoC was tested on the Virtex-4 IPD FPGA test board manufactured by Hanback Electronics. Of the FPGA resources available to the Virtex-4 device, the proposed architecture utilized 57% of the 768 available IO pins representing 7.42%. It also utilized 1 out of the 32 available global buffers, 27387 out of the 71680 available LUTs, and 13694 out of the 35840 available CLB Slices, both representing 38.21%. Not all, the Register utilization for the proposed integrated SoC was 26.76% representing 19185 out of the available 71680. Additionally, the architecture utilized 31 of the 200 Block RAMs. The proposed SoC architectures operation was verified on the Virtex-4 FPGA board by generating derived keys using the ECDH + HKDF core and encrypting the standard Lena image file (128x128) using the AEAD core. The proposed architecture operates at a maximum frequency of 273 MHz. Regarding subsequent work, the architectures will further be tested on two different boards, each acting and sender and receiver respectively and exchanging communication over Bluetooth protocol.

REFERENCES

[1] "Number of IoT devices 2015-2025," *Statista*. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (accessed May 08, 2022).
[2] "Five ways IoT can make your life easier." https://www.metrikus.io/blog/five-ways-iot-can-make-your-life-easier (accessed May 08, 2022).
[3] "The 3 Biggest Factors in IoT Technology Success." https://www.samsungsds.com/la/insights/IoT-success-factors-eng.html (accessed May 08, 2022).
[4] "Top Cybersecurity Threats in 2021," *University of San Diego Online Degrees*, Sep. 13, 2016. https://onlinedegrees.sandiego.edu/top-cyber-security-threats/ (accessed May 08, 2022).
[5] M. Bellare and P. Rogaway, "Minimizing the use of random oracles in authenticated encryption schemes," in *Information and Communications Security*, Berlin, Heidelberg, 1997, pp. 1–16. doi: 10.1007/BFb0028457.
[6] M. Abdalla, M. Bellare, and P. Rogaway, "DHAES: An Encryption Scheme Based on the Diffie-Hellman Problem," Feb. 1970.
[7] M. Abdalla, M. Bellare, and P. Rogaway, "The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES," in *Topics in Cryptology — CT-RSA 2001*, Berlin, Heidelberg, 2001, pp. 143–158. doi: 10.1007/3-540-45353-9_12.
[8] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985, doi: 10.1109/TIT.1985.1057074.
[9] American National Standards Institute, "ANSI X9.63, (2001). Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography," Nov. 2001, [Online]. Available: https://standards.globalspec.com/std/26827/X9.63
[10] "IEEE Standard Specifications for Public-Key Cryptography - Amendment 1: Additional Techniques," *IEEE Std 1363a-2004 (Amendment to IEEE Std 1363-2000)*, pp. 1–167, Sep. 2004, doi: 10.1109/IEEESTD.2004.94612.
[11] V. G. Martínez, F. H. Álvarez, L. H. Encinas, and C. S. Ávila, "Analysis of ECIES and Other Cryptosystems Based on Elliptic Curves," p. 9.
[12] G. Kanda, A. O. A. Antwi, and K. Ryoo, "Hardware Architecture Design of AES Cryptosystem with 163-Bit Elliptic Curve," in *Advanced Multimedia and Ubiquitous Engineering*, Singapore, 2019, pp. 423–429. doi: 10.1007/978-981-13-1328-8_55.
[13] G. Kanda and K. Ryoo, "Efficient Implementation of Digital Standard Cells-Based True Random Number Generator for Securing FPGA Designs," *TEST Engineering & Management*, vol. 83, pp. 3996–4007, Mar. 2020.
[14] G. Kanda and K. Ryoo, "High-Throughput Low-Area Hardware Design of Authenticated Encryption with Associated Data Cryptosystem that Uses ChaCha20 and Poly1305," *IJRTE*, vol. 8, no. 2S6, pp. 86–94, Sep. 2019, doi: 10.35940/ijrte.B1017.0782S619.
[15] *PicoRV32 - A Size-Optimized RISC-V CPU*. Yosys Headquarters, 2022. Accessed: May 08, 2022. [Online]. Available: https://github.com/YosysHQ/picorv32
[16] "Elliptic-curve Diffie–Hellman," *Wikipedia*. Apr. 29, 2022. Accessed: May 08, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Elliptic-curve_Diffie%E2%80%93Hellman&oldid=1085310059
[17] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976, doi: 10.1109/TIT.1976.1055638.
[18] A. Rukhin *et al.*, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," p. 131.
[19] D. J. Bernstein, "CubeHash specification (2.B.1)," p. 4.
[20] "Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family," *Federal Register*, Nov. 02, 2007. https://www.federalregister.gov/documents/2007/11/02/E7-21581/announcing-request-for-candidate-algorithm-nominations-for-a-new-cryptographic-hash-algorithm-sha-3 (accessed May 08, 2022).
[21] H. Krawczyk, "Cryptographic Extraction and Key Derivation: The HKDF Scheme," in *Advances in Cryptology – CRYPTO 2010*, vol.

International Journal of
Electrical and Electronics Research (IJEER)
Research Article | Volume 10, Issue 2 | Pages 230-244 | e-ISSN: 2347-470X

6223, T. Rabin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 631–648. doi: 10.1007/978-3-642-14623-7_34.
[22] D. J. Bernstein, "ChaCha, a variant of Salsa20," p. 6.
[23] D. J. Bernstein, "The Poly1305-AES Message-Authentication Code," in *Fast Software Encryption*, Berlin, Heidelberg, 2005, pp. 32–49. doi: 10.1007/11502760_3.
[24] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," Internet Engineering Task Force, Request for Comments RFC 7539, May 2015. doi: 10.17487/RFC7539.
[25] A. O. Freier, P. Karlton, and P. C. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0," Internet Engineering Task Force, Request for Comments RFC 6101, Aug. 2011, doi: 10.17487/RFC6101.
[26] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," *J Cryptol*, vol. 21, no. 4, pp. 469–491, Oct. 2008, doi: 10.1007/s00145-008-9026-x.
[27] J. Gorban, "UART IP Core Specification," p. 18.
[28] "Implementation of a secure TLS coprocessor on an FPGA," *Microprocess. Microsyst.*, vol. 40, no. C, pp. 167–180, Feb. 2016, doi: 10.1016/j.micpro.2015.10.009.
[29] T. David, B. Johan, and C. Lin, (2021), "Research on Real-time Data Transmission between IoT Gateway and Cloud Platform based on Two-way Communication Technology," *International Journal of Smartcare Home*, vol. 1, no. 1, pp. 61-74, Jun. 2021.
[30] I. S. Fathi, M. A. Ahmed, M. A. Makhlouf, and E. A. Osman, "Compression Techniques of Biomedical Signals in Remote Healthcare Monitoring Systems: A Comparative Study," *International Journal of Hybrid Information Technologies*, vol. 1, no. 1, pp. 33-50, Sep. 2021, doi: 10.21742/IJHIT.2021.1.1.03.
[31] S. Y. Lee, "Blockchain-based Medical Information Sharing Service Architecture," International Journal of IT-based Public Health Management, vol. 8, no. 1, pp.27-32, Sep. 2021, doi: 10.21742/IJIPHM.2021.8.1.04.
[32] S. A. Alhumrani and Jayaprakash Kar, "Cryptographic Protocols for Secure Cloud Computing", International Journal of Security and Its Applications, NADIA, ISSN: 1738-9976 (Print); 2207-9629 (Online), vol.10, no.2, February (2016), pp. 301-310, http://dx.doi.org/10.14257/ijsia.2016.10.2.27.