Integration, the VLSI Journal 68 (2019) 108–121

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

FPGA implementation of dynamically reconfigurable IoT security module using algorithm hopping

Shady Soliman [a], Mohammed A. Jaela [a], Abdelrhman M. Abotaleb [d], Youssef Hassan [d], Mohamed A. Abdelghany [a,b], Amr T. Abdel-Hamid [a], Khaled N. Salama [c], Hassan Mostafa [d,e,*]

[a] Electronics Department, German University in Cairo, Cairo 11835, Egypt
[b] Integrated Electronic Systems Lab, GTU Darmstadt, Germany
[c] Computer, Electrical and Mathematical Sciences and Engineering (CEMSE) Division, King Abdullah University of Science and Technology (KAUST), Thuwal 23955-6900, Saudi Arabia
[d] Electronics and Communications Engineering Department, Cairo University, Giza 12613, Egypt
[e] University of Science and Technology, Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, October Gardens, 6th of October, Giza 12578, Egypt

ARTICLE INFO

Keywords:
CAESAR
FPGA
DPR
Cryptography
Hopping
AEAD
IoT

ABSTRACT

Internet of Things (IoT) is a promising technology that is continuously spreading around the world leading to many challenges facing cryptographic designers who are trying to fulfill the security standards of IoT constrained devices. In this work, a new design is proposed that adds a new dimension of security by using the concept of frequency hopping to generate a pseudo-random pattern for switching between 5 lightweight cryptographic ciphers: AEGIS, ASCON, COLM, Deoxys and OCB that are participating in the Competition for Authenticated Encryption, Security, Applicability, and Robustness (CAESAR). The proposed design exploits the advantages of Dynamic Partial Reconfiguration (DPR) technology in Field Programmable Gate Arrays (FPGAs) to switch between the 5 ciphers using Internal Configuration Access Port controller (AXI-HWICAP) providing a decrease of 58% and 80% in area utilization and power consumption respectively. The design is synthesized using Xilinx Vivado 2015.2 and mounted on Zynq evaluation board (XC7Z020LG484-1).

1. Introduction

Internet of Things (IoT) is a network of devices connected to each other in a wired or non-wired way where each device has a unique identity. These devices process data and send it to each other without human intervention [1]. The term (IoT) usually refers to resource-limited objects such as sensors, RFID tags or any other contactable device that has the ability to compute data while connected to the internet [2]. IoT is spreading rapidly and the number of connected devices is expected to reach 18 Billion by 2022 [2]. However, concerns about privacy and security are increasing, especially given that IoT takes considerable place in the contexts of governments and organizations, as well as infrastructure of public institutions. To address this issue, lightweight cryptography is becoming a considerable approach to make the connection and transfer of data between constrained devices more secure [3].

Lightweight cryptography is a cryptographic algorithm or protocol tailored for implementation in constrained environments such as RFID tags, sensors, contactless smart cards, and health-care devices [4]. One of the most effective ways of exploiting lightweight cryptographic algorithms is by using authenticated encryption schemes. Authenticated encryption schemes are a class of symmetric key cryptographic algorithms that ensure that both of confidentiality and authenticity of data are provided at the same time [5]. Confidentiality includes protecting data from being exposed without permission, while authenticity comprises ensuring both integrity of data and verification of its

* Corresponding author. University of Science and technology, Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, October Gardens, 6th of October, Giza 12578, Egypt.

E-mail addresses: shady_soliman2002@yahoo.com (S. Soliman), morhaf.jaela@gmail.com (M.A. Jaela), aabotaleb@nu.edu.eg (A.M. Abotaleb), youssef_hassan_gamal@hotmail.com (Y. Hassan), mohamed.abdel-ghany@guc.edu.eg (M.A. Abdelghany), amr.talaat@guc.edu.eg (A.T. Abdel-Hamid), khaled.salama@kaust.edu.sa (K.N. Salama), hmostafa@uwaterloo.ca (H. Mostafa).

https://doi.org/10.1016/j.vlsi.2019.06.004
Received 19 January 2019; Received in revised form 2 May 2019; Accepted 16 June 2019
Available online 26 June 2019

source. Sometimes data such as packet headers are required for handling by some applications, which requests authentication without the need for encryption. Such schemes are usually defined as Authenticated Encryption with Associated Data (AEAD) which is embraced in this work. AEAD scheme provides one level of data security which is the symmetric key. However, nothing beyond this level has ever been added to tighten the security of data against attacks due to limited resources used by IoT devices. In this work, a new design is proposed to introduce a second dimension of security using the concept of algorithm hopping by switching among five lightweight cryptographic algorithms that are currently under review in the ongoing CAESAR competition. The switching is performed using dynamic partial reconfiguration (DPR) technology to maintain a reasonable power consumption and area utilization, taking into consideration the limited resources of IoT devices while adding the second dimension of security based on the pseudo-random pattern for switching among different algorithms.

2. CAESAR

CAESAR is an ongoing contest for Authenticated Encryption, Security, Applicability and Robustness. The main target of the contest is to find new and improved versions of the AEAD schemes [6]. CAESAR started in 2014 with 54 initial submissions in the first round. The contest is at the fourth and final round at the time of creating the design proposed in this work. Five out of seven finalists are chosen for the proposed design: AEGIS, ASCON, COLM, Deoxys and OCB. The selection of these 5 algorithms is a case study to show the effectiveness of using the proposed algorithm hopping technique and this technique is expendable to accommodate any other security algorithms.

2.1. AEGIS

AEGIS is a dedicated authenticated encryption algorithm where a message is used to update the state of the cipher, and message authentication is achieved almost for free [7]. AEGIS is constructed from the Advanced Encryption Standard (AES) round function. It has three variations: AEGIS-128L, AEGIS-128 and AEGIS-256. The first one uses eight AES round functions to process a 32-byte message block, the second one processes a 16-byte message block with five AES round functions and the third one uses six AES round functions. AEGIS is very fast and its computational cost is half that of AES [7]. AEGIS also offers high security as long as the initialization vector is not reused, which makes it impossible to recover the AEGIS state and key faster than exhaustive key search. AEGIS is suitable for network communication since it can protect a packet while leaving the packet header (associated data) unencrypted. The state update function of AEGIS-128 updates the 80-byte state S_i with a 16-byte message block m_i. S_{i+1} = StateUpdate128 (S_i, m_i) is given as follows:

$$S_{i+1,0} = AESRouund(S_{i,4}, S_{i,0} \oplus m_i) \quad (1)$$

$$S_{i+1,1} = AESRouund(S_{i,0}, S_{i,1}) \quad (2)$$

$$S_{i+1,2} = AESRouund(S_{i,1}, S_{i,2}) \quad (3)$$

$$S_{i+1,3} = AESRouund(S_{i,2}, S_{i,3}) \quad (4)$$

$$S_{i+1,4} = AESRouund(S_{i,3}, S_{i,4}) \quad (5)$$

The state update function is shown in Fig. 1:

2.2. ASCON

ASCON presents a low hardware scheme with extremely low memory requirements. While the scheme provides full security of 128 bits, it offers no nonce misuse resistance [8]. It is based on the Sponge wrap/Monkey Duplex mode of operation and eliminating inverse operations [9]. ASCON is optimized for minimal overhead (cipher text = plain text) and it is lightweight for constrained devices making it fast in hardware and software implementations. ASCON has several parameters used for encryption such as: secret key (K), associated data (A), public message number that is denoted by nonce (N), and Initialization Vector (IV), in order to encrypt a plaintext (P), according to the formula:

$$E_{a,b,k,r}(K, N, A, P) = (C, T) \quad (6)$$

where a, b are the numbers of permutation rounds, r is the state size, and k is the secret key size. The output of this process is the ciphertext C, and the authentication tag T [8].

Fig. 2 illustrates ASCON modes of operations which are the encryption mode and the decryption mode. P block is the main block in ASCON algorithm. P block has two flavors: one for carrying out the initialization/finalization process (Pa) and the other for performing the internal processes (Pb).

2.3. COLM

COLM [10] is a block cipher based on Encrypt-Linear mix-Encrypt mode, designed with the goal to achieve online misuse resistance, to be fully parallelizable, and to be secure against blockwise adaptive adversaries.

The authenticated encryption for complete message block is shown in Fig. 3. COLM consists of two-layer parallelizable encryption. COLM mixes the output of the first encryption layer to generate the input to the second encryption layer, using linear mixing function. The high-speed COLM implementation instantiates two instances of AES to implement the two layers of encryption. In order to optimize COLM for low area, only one instance of AES is used to perform the two encryption layers.

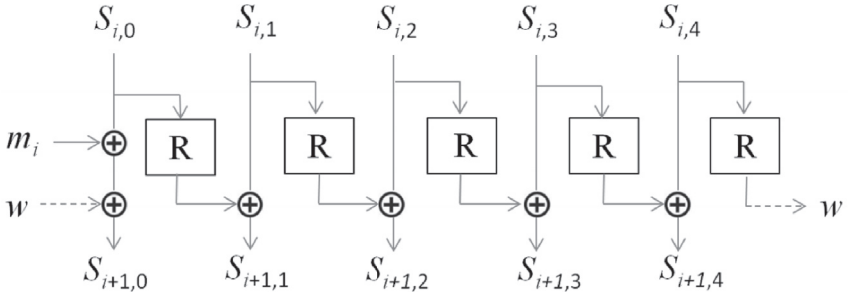

Fig. 1. The state update function of AEGIS-128. R indicates the AES encryption round function without XORing with the round key and w is a temporary 16-byte word [7].

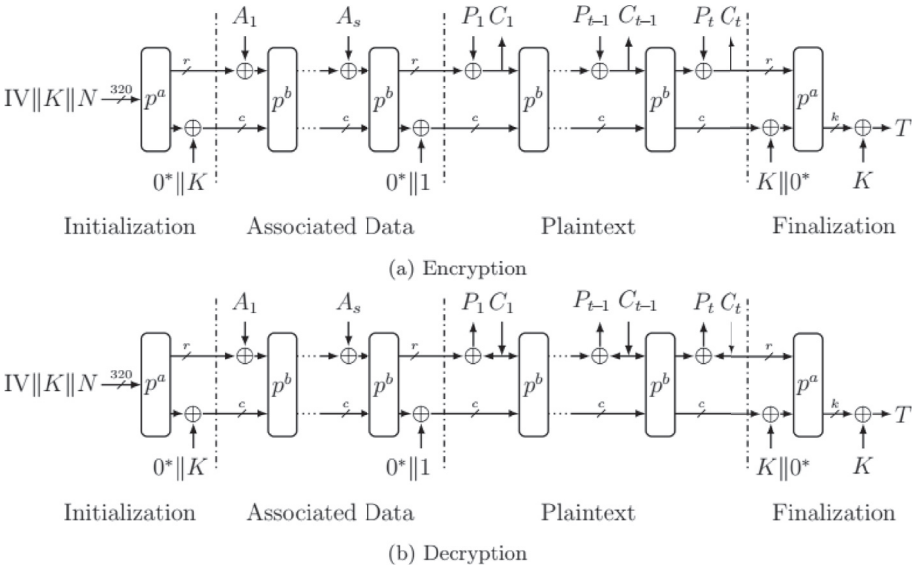

Fig. 2. ASCON's mode of operation [8].

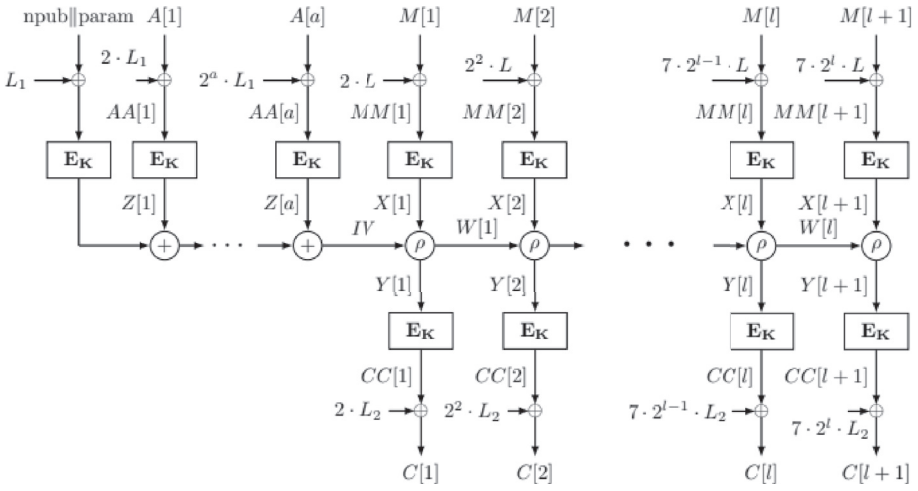

Fig. 3. COLM authenticated encryption for complete message block. E_K. denotes the block cipher AES-128 [10].

A Finite state machine and Multiplexers are added to control the data flow to the AES. The optimized encryption operation is processed in twice the clock cycles of the non-optimized one and the same applies for the decryption operation.

2.4. Deoxys

Deoxys presents a new authenticated encryption design based on custom made tweakable block ciphers using the AES round function as a building block [11]. Deoxys is an authenticated encryption scheme that provides full 128-bit security for both privacy and authenticity making it efficient in software. Moreover, Deoxys performs particularly well for small messages (only m + 1 block cipher calls are required for an m block message and no precomputation is required). In the nonce-misuse resistant versions of Deoxys, in addition to a full 128-bit security for unique nonces, birthday-bound security is obtained when the nonce is reused. Finally, Deoxys can be lightweight and the key can be hardcoded for further smaller area footprint. Deoxys uses a tweakable block cipher Deoxys-BC as internal primitive. Deoxys has two main mode variants:

- Nonce-Respecting Mode: this variant is for where adversaries are assumed to be nonce-respecting, meaning that the user must ensure that the value N will never be used for encryption twice with the same key.
- Nonce-Misuse Resistant Mode: this variant is a new authenticated encryption mode named SCT, relaxes this constraint and allows the user to reuse the same N with the same key.

In this work, the focus is only on the second mode. The encryption algorithm is depicted in Figs. 4 and 5 for the authentication part and in Fig. 6 for the encryption part.

2.5. OCB

OCB (short for Offset Codebook) is an AEAD scheme that depends on a block cipher that must have a 128-bit block size [12]. OCB achieves both confidentiality and authenticity. Confidentiality is defined such that an adversary is unable to distinguish OCB-outputs from an equal number of random bits, while authenticity means that an adversary

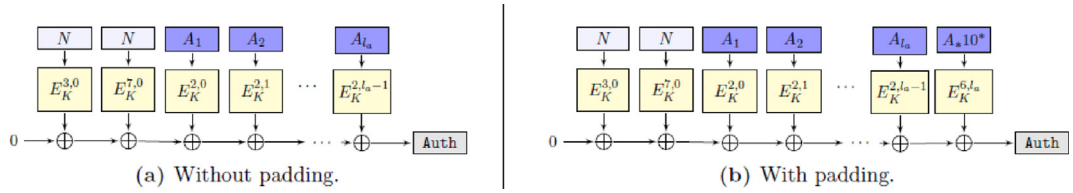

Fig. 4. Handling of the associated data for the nonce-misuse resisting mode: in the case where the associated data is a multiple of the block size, no padding is needed [11].

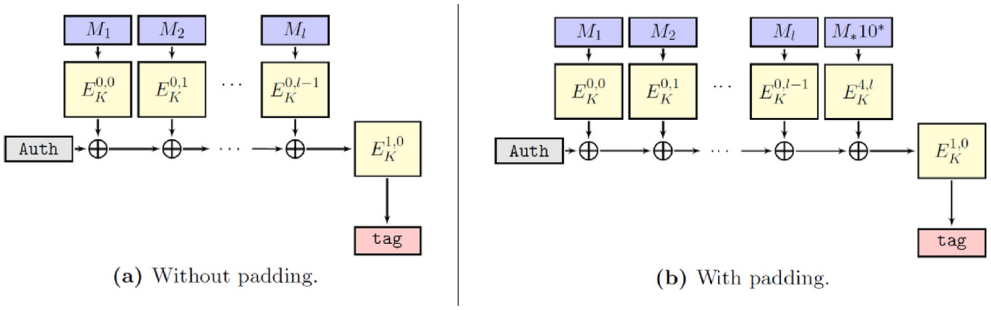

Fig. 5. Message processing in the authentication part of the nonce-misuse resisting mode: in the case where the message-length is a multiple of the block size, no padding is needed [11].

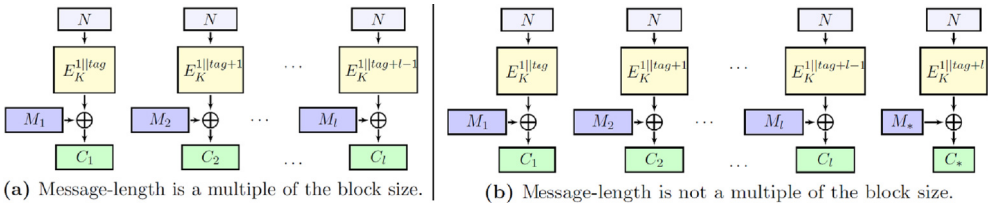

Fig. 6. Message processing for the encryption part of the nonce-misuse resisting mode [11].

is unable to produce any valid nonce-cipher text pair that it has not already acquired. OCB is nearly as fast as Counter mode (CTR) because each block encryption requires just a few xors on top of an AES call. OCB is parallel as most of the computations are independent of one another which allows both hardware and software accelerations. It is also designed for minimal authentication overhead beyond what is required for provable security and simple encryption using a block cipher. OCB is not designed to resist nonce reuse or to enjoy beyond birthday bound security [12]. Fig. 7 is the illustration of OCB[E, τ]. Here E: $\kappa \times \{0,1\}^{128} \rightarrow \{0,1\}^n$ is a blockcipher and $\tau \in [0 \ldots 128]$ is the tag length. $M, C \in \{0, 1\}^*$. In the top section of the figure, Message M has a full final block ($|M4 = n|$) (Checksum $= M_1 \oplus M_2 \oplus M_3 \oplus M_4$). In the middle section, Message M has a short final block, $1 \leq |M_*| < n$ (Checksum $= M_1 \oplus M_2 \oplus M_3 \oplus M_*10^*$). In the bottom side, An AD of three full blocks (left) or two full blocks and one short one (right). Throughout: Offsets (the Δ-values) are updated and used top-to-bottom, then left-to-right. Offset initialization and update functions (Init, Inc_i, $\text{Inc}_\$$, Inc*) return n-bit strings. Each flavor of increment is a xor with some precomputed, K-dependent value [12].

3. Algorithm hopping

Algorithm hopping is analogous to the Frequency Hopping Spread Spectrum (FHSS) which is an approach to send radio signals by rapid switching of carriers between many frequency channels. The switching is based on a pseudo-random pattern only known to the transmitter and the receiver [13]. It is widely used as a multiple access method in the Code Division Multiple Access Scheme (CDMA) in communication systems. The proposed design uses this idea by modeling the frequency channels as the encryption algorithms while sending the block data to be encrypted/decrypted as an equivalent to sending radio signals over different frequency channels. Normally, any cryptography systems uses one algorithm all the time and only the key is changeable during the run time. In the proposed algorithm hopping, the algorithm will be change per session. Also the key can be changed at any time. Therefore, if the attacker needs to hack the proposed design, the number of algorithms used in addition to the ID of the each algorithm for each session are required for hacking. After that the attacker needs to know the key to the used algorithm. This means that the proposed design is more secure than the current designs. Fig. 8 shows the proposed algorithm hopping technique, the design hops among the five AEAD schemes that are selected from the CAESAR competition. For example, during session 2, the algorithm that is used is COLM and for session 4, the AEAD scheme is OCB.

The sequence of switching is generated using the Linear Feedback Shift Register (LFSR). LFSR is implemented as a series of Flip-Flops, wired together as a shift register [14]. Some taps of the shift register chain are used as inputs to either an XOR or XNOR gate. The output of this gate is then used as a feedback to the beginning of the shift register chain. There are some special properties of the LSFR that can be listed as follows:

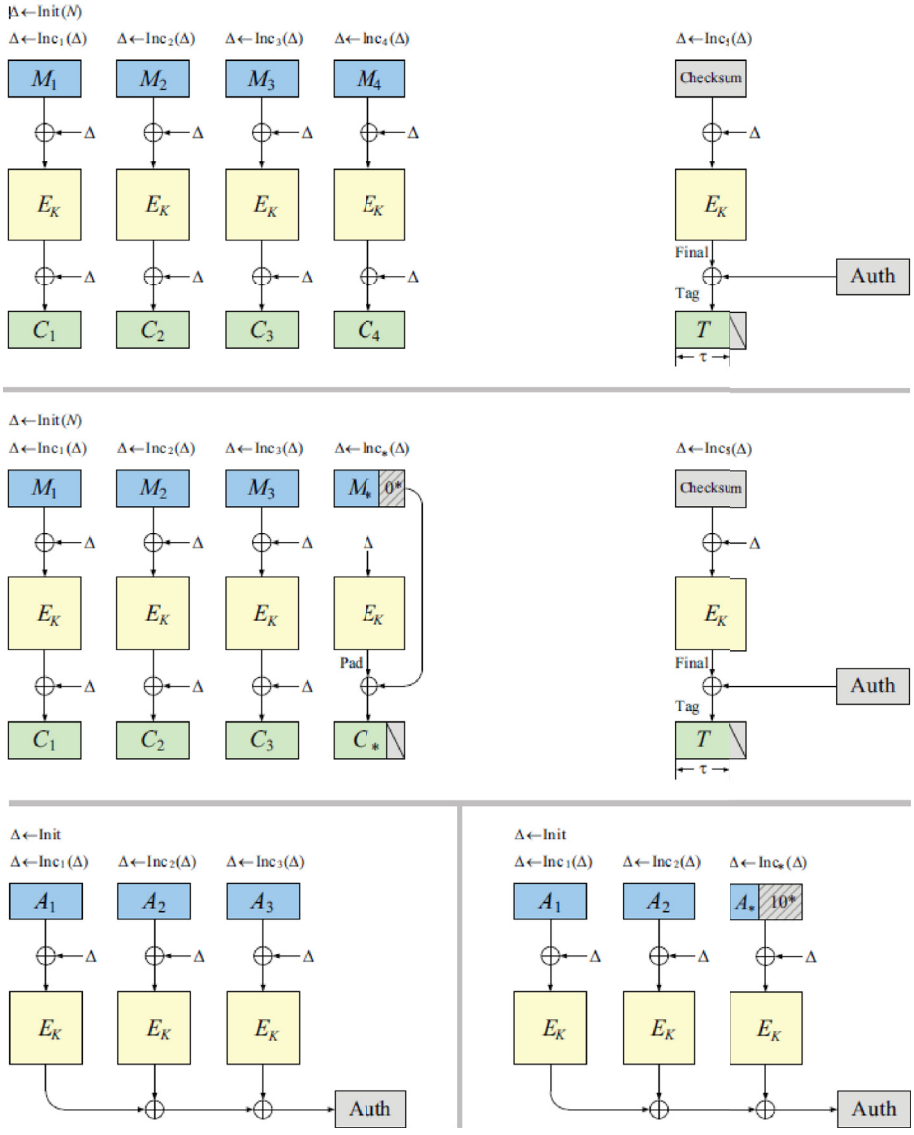

Fig. 7. Illustration of OCB [12].

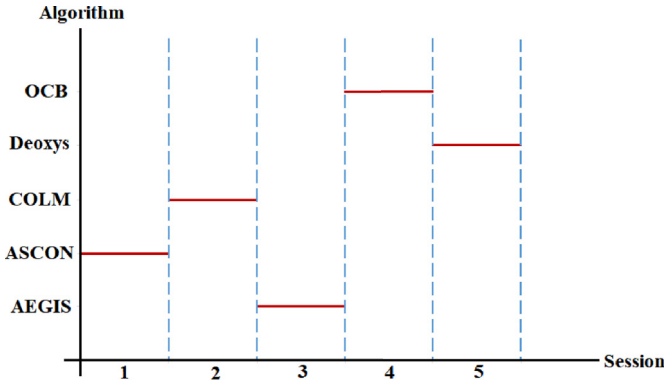

Fig. 8. Proposed Algorithm Hopping technique.

- LFSR patterns are pseudo-random.
- Output patterns are deterministic. The next state can be figured out by knowing the position of the XOR gates as well as the current pattern.
- A pattern of all 0's cannot appear when the taps use XOR gates.

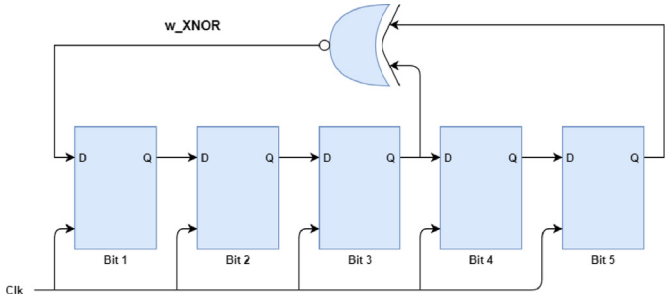

Fig. 9. 5-bit LSFR using XNOR gate.

- A pattern of all 1's cannot appear when the taps use XNOR gates.
- The maximum possible number of iterations of any LFSR:

$$= 2^m - 1 \tag{7}$$

m: number of bits which is the number of Flip-Flops

An example design of a 5-bit LSFR using XNOR gate is shown in Fig. 9.

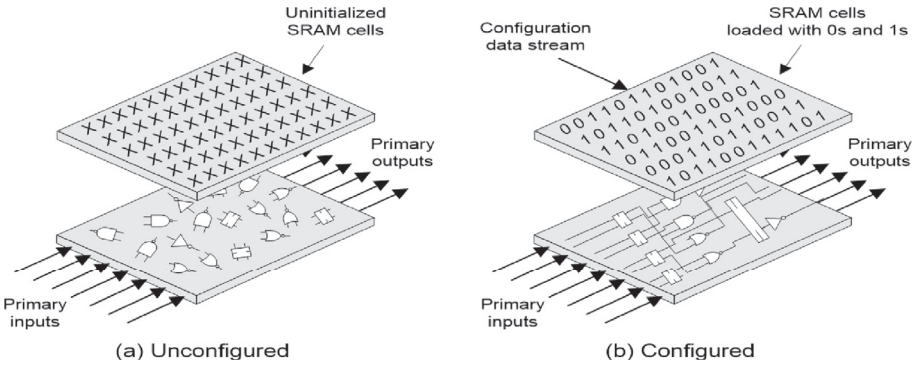

Fig. 10. Two distinct layers of FPGA [15].

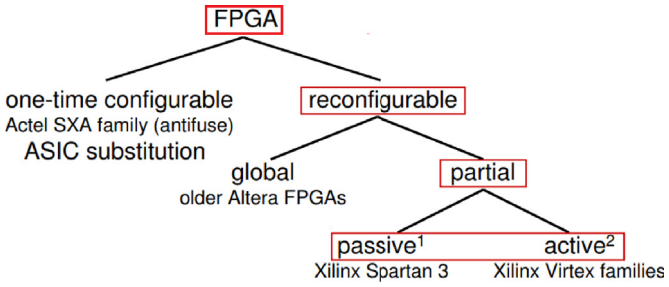

Fig. 11. Classification of FPGAs by their configuration capabilities [15].

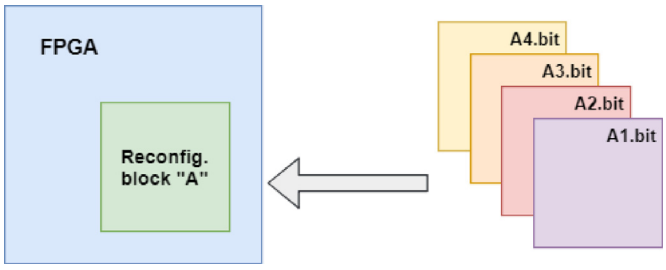

Fig. 12. Basic structure of partial reconfiguration design.

4. Dynamic partial reconfiguration

4.1. FPGA configuration

FPGAs contain a large amount of programmable logic and registers, which are connected together in different ways to realize different functions. It is sometimes useful to imagine that the FPGA consists of two distinct layers: the logic gates/registers and the programmable SRAM configuration cells (Configuration Memory (CM)) [15] as shown in Fig. 10.

An FPGA is reconfigured by writing bitstream into Configuration Memory (CM) that controls function computed on logic layer. FPGA architectures can be categorized according to the capability of the configuration, as illustrated in Fig. 11. At the top level, FPGAs are divided into one-time configurable devices that can only be applied as an ASIC substitute and configurable FPGAs [16]. Configurable FPGA devices are categorized partially and globally reconfigurable devices [15]. In the globally reconfiguration category, the whole chip configuration is swapped. Consequently, all the states inside the FPGA are overwritten. However, in partially reconfigurable systems, some states of the FPGA chip are updated. Partial reconfiguration is achieved either passive by reconfiguring a portion of the device (changing the functionality) when the device is inactive without affecting other areas of the device or active where the operation can seamlessly continue during the reconfiguration process [17].

4.2. Technology

The original idea of FPGAs is to provide on-site programming and reprogramming without the need to re-fabricate the modified logic by loading several bit files corresponding to different designs. DPR improves the original idea by allowing for the modification of an operating FPGA design by loading partial bit files after configuring the device with an initial full bit file [18]. This requires the designer to divide his design into 2 parts: Static part and dynamic part. The

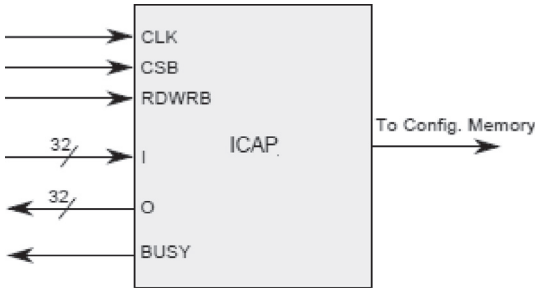

Fig. 13. Xilinx ICAP primitive [21].

dynamic part consists of a set of Reconfigurable Modules (RMs) which are swapped during runtime after being floor-planned onto a Reconfigurable Partition (RP). When it is desired to switch to a certain RM, its corresponding partial bit file is loaded at runtime without affecting the static part. A block diagram illustrating different parts of a DPR dependent design is shown in Fig. 12.

4.3. DPR controllers

Currently, there are several DPR controllers implemented as Intellectual Properties (IPs) and offered by Xilinx such as: HWICAP, PRC and PCAP [18]. However, it is possible to implement custom IP controllers such as ZYCAP [19]. A comparative study was done in Ref. [20] showing the performance of these controllers based on area utilization, power consumption and maximum throughput for a software defined radio encoder. From Ref. [20], it is stated that PRC consumed most resources while providing the highest throughput in comparison with HWICAP which consumed much less resources while providing the least throughput.

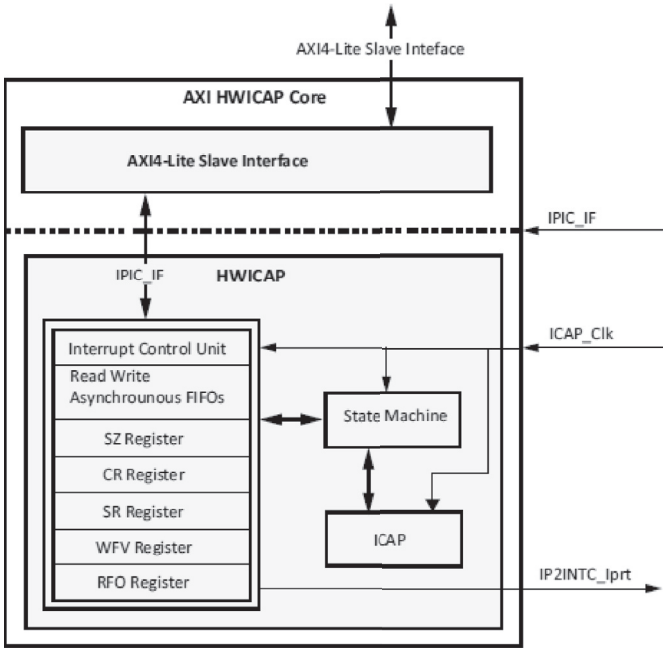

Fig. 14. Top level block diagram for the AXI HWICAP core [21].

4.4. Xilinx AXI-HWICAP controller

Xilinx provides many IP cores to connect the ICAP module with the user design. These IP cores corresponds to partial reconfiguration controller that enables an embedded microprocessor such as ARM processors or Microblaze to access the configuration memory. ICAP is a xilinx predefined macro that has direct access to the configuration memory for both write and read modes [18] as depicted in Fig. 13. CSB is the active low interface select signal, RDWRB is the read/write select signal. BUSY is valid only for read operations and remains low for write operations. In Xilinx 7-series FPGAs, the ratio of the ICAP interface data width to the configuration memory is 32 bit wide. The max theoretical reconfiguration throughput of ICAP is equal to 400 MB/s at a frequency of 100 MHz [21]. In practice, it was found that the measured throughput is much less than the theoretical one due to the addition of the reconfiguration overhead to the DPR at the system level [22].

AXI-ICAP [21] is an ICAP controller designed for AXI bus interfaces where it is connected as a slave peripheral. During DPR, the partial bitstream data are buffered from an external memory to a write/read FIFOs inside the core where there is a finite state machine that monitors the status of the FIFOs and supplies partial bitstream data to the ICAP and then to the configuration memory as displayed in Fig. 14. Based on the study in Ref. [20], AXI-ICAP is the optimum choice as a controller in the proposed design despite the limitations on the maximum throughput which is not an issue for IoT power-constrained devices that requires low area and power requirements.

5. Proposed design

5.1. Design modules

As shown in Fig. 15, the design is divided into 2 parts: Encryption module and decryption module. Each module is loaded to an FPGA and accordingly, each module has 2 parts: Static part and dynamic part. It is important to note that both parts include the same hardware modules, LFSR is used at both the encryption and decryption sides to make both sides working on the same ciphering algorithm while hopping.

The static part for both modules consists of the following components:

- Universal Asynchronous Receiver Transmitter (UART).
- First Word Fall Through First In First Out (FWFT-FIFOs) for inputs and outputs.
- AEAD top.

The dynamic part for both modules consists of the following components:

- Pre processor.
- Cipher core.
- Post processor.

The functionality of each component is explained as follows:

5.1.1. AEAD top

During CAESAR contest, a universal Application Programming Interface (API) is created to reduce any potential biases in benchmarking of participating authenticated ciphers in hardware [23]. This API named AEAD has the following features:

- Wide range of data port widths ranging from 8-bits to 256-bits.
- Independent data and key inputs.
- Simple high-level communication protocol.
- Support for encryption and decryption within the same core.
- Ability to communicate with FIFOs.

As shown in Fig. 16, the AEAD interface consists of 3 main data buses:

- Public Data Inputs (PDI)
- Secret Data Inputs (SDI)
- Data Outputs (DO)

In addition, there are several control signals such as: valid and ready. The valid signal indicates that the source is ready to send data

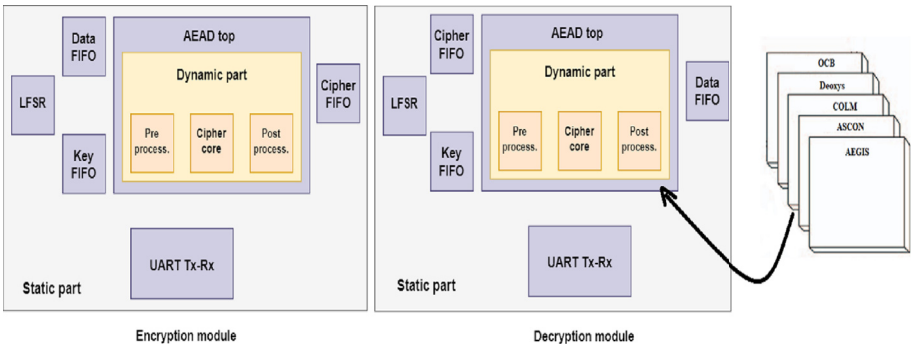

Fig. 15. Proposed design.

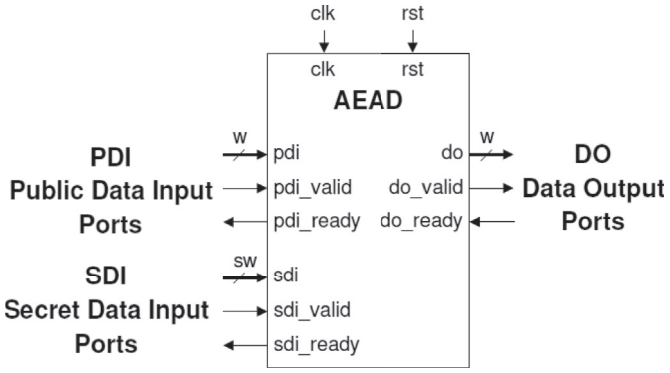

Fig. 16. AEAD interface [23].

while the ready signal indicates that the destination is ready to receive them. All possible inputs and outputs of the AEAD are illustrated in Fig. 17. AD denotes Associated Data, Npub denotes Public Message Number, such as Number used once (Nonce) or Initialization Vector. Nsec denotes Secret Message Number, which was recently introduced in some authenticated ciphers. Both Npub and Nsec are expected to be unique for each message encrypted using a given key. The difference is that Npub is directly buffered to the other side, while Nsec is sent in the encrypted form. SDI port is responsible for processing the secret key while PDI port handles the rest of the inputs.

The API is also composed of several generic parameters such as: Key length, data block size, SDI port width, PDI port width and type of data padding which are modified based on the requirements of each cipher. The main idea of the AEAD top is to modify the API by choosing the largest of the generic parameters of the 5 ciphers and grouping them in 1 static port from which each dynamic module's parameters are modified.

5.1.2. LFSR

The main purpose of the LFSR is to generate a pseudo-random pattern based on an input seed by the user at run-time that controls the start of the sequence of IDs assigned to each algorithm. Since DPR allows using any number of algorithms, it is also possible to expand the size of the LFSR to generate a longer pattern with more IDs to accommodate for the increasing number of algorithms used. A Fibonacci LFSR of size = 5-bit with feedback polynomial $1 + x^3 + x^5$ is selected to generate $2^5 - 1$ pseudo-random numbers. The IDs of the algorithms to be used in the hopping sequence are extracted from the LFSR's least three significant bits. For our system to operate, few assumptions and considerations are made.

1. Assumptions

   (a) LFSRs at both ends are synchronized to run over the same hopping sequence.

   (b) If least 3-bits of LFSR ≥4 then the selected algorithm to be used by DPR will be the AEGIS.

2. Considerations

   (a) LFSR size will affect the randomization pool and repetition of the random hopping sequence, so it is mainly controlling the hopping pattern rate.

   (b) Initialization of the LFSR is the critical word, once known the random hopping sequence can be tracked easily, so it must be exchanged securely same as the symmetric key.

5.1.3. FWFT FIFO

The first word fall through FIFOs is used for data buffering with ability of customizing the data size and depth to accommodate for different data block sizes for the different algorithms used in the proposed algorithm hopping technique. First word fall through allows for reading data written without the need to assert a strobe for reading request, which results in better performance.

5.1.4. UART

As shown in Fig. 15, a complete UART is implemented in both encryption/decryption sides. It is responsible for sending and receiving encryption/decryption data between the 2 modules. Moreover, it sends feedback signals to the encryption module to initiate the next hop in sequence.

The proposed security module considers the UART as a separate block for communications with the embedded processor, which is typical for IoT security.

5.1.5. Pre-processor and post-processor

The pre-processor is responsible for the execution of the following tasks common for all used algorithms:

- Parsing segment headers
- Loading and activating keys
- Serial-In-Parallel-Out loading of input blocks
- Padding input blocks
- Keeping track of the number of data bytes left to process

The post-processor is responsible for the following tasks:

- Clearing any portions of output blocks not belonging to cipher or plaintext
- Parallel-In-Serial-Out conversion of output blocks into words
- Formatting output words into segments
- Storing decrypted messages until the result of authentication is known
- Generating the status block with the result of authentication

The pre-processor and post-processor cores are highly configurable using generics. These generics can be used to determine:

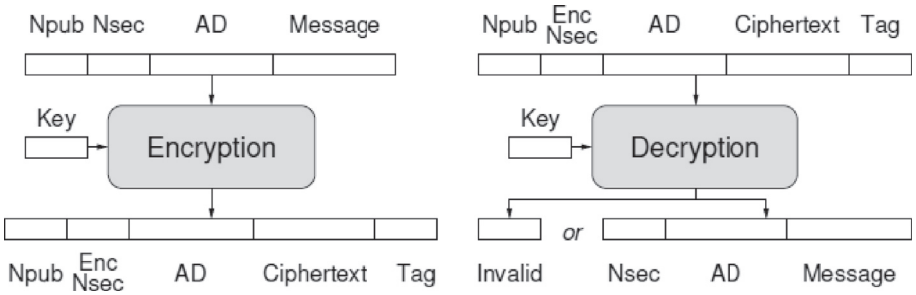

Fig. 17. Inputs and outputs of AEAD [23].

```
# 001 : Instruction(Opcode=Load Key)
INS = 0104010000000000
# 001 : SgtHdr (Size=  16) (EOI=1)(EOT=1)(SgtType=Key)
HDR = 0163000000000010
DAT = D7B1CB5221D16D92
DAT = BB910D157C6F1C04
```

Fig. 18. Load key instruction.

```
# 001 : Instruction (Opcode=Activate Key)
INS = 0105010000000000
```

Fig. 19. Load key instruction.

- The widths of the pdi, sdi, and do ports
- The size of the message/ciphertext block, key, nonce, and tag
- Padding for the associated data and the message
- Types and order of segments expected by a particular cipher

For the first message and the subsequent key change, a new key must be loaded into the pre-processor via the SDI port first. This can be done by providing the Load Key instruction. A typical key loading sequence of words is shown in Fig. 18. In this example, the first word specifies the Load Key instruction. The second word specifies that the subsequent data segment is of the key type, with the size of 16 bytes (128 bits). This segment is also the end-of-type and the end-of-input segment. The next two words consist of the data representing the key.

Before the new key becomes active, it must be activated via the PDI port first. This mechanism facilitates the synchronization between the two input ports. It also allows loading a new key without interfering with the key that is being used. A typical key activation process is shown in Fig. 19. This word must be applied before any other instruction word. Loading of round keys precomputed in software can be performed in a similar way, with the instruction Load Key replaced by Load Round Key, followed by a segment composed of a sequence of round keys.

This word must be applied before any other instruction word. Loading of round keys precomputed in software can be performed in a similar way, with the instruction Load Key replaced by Load Round Key, followed by a segment composed of a sequence of round keys.

5.1.6. Cipher core

The development of the Cipher core is left to individual designers and can be performed using their own preferred design methodology. Typically, when using a traditional RTL (Register Transfer Level) methodology, the Cipher core data path is first modeled using a block diagram, and then translated to a hardware description language (VHDL or Verilog HDL). The Cipher core Controller is then described using an algorithmic state machine (ASM) chart or a state diagram, further translated to HDL.

The algorithmic state machine (ASM) of the Cipher core Controller is typically characterized by the following groups of states:

1. Load and/or activate the key
2. Process the associated data
3. Process the message/ciphertext
4. Generate/verify an authentication tag

In the first group of states, load and activate the key, the cipher core should monitor the key_needs_update and key_ready inputs, and provide key_updated output at the appropriate time. The circuit should operate as follows:

After reset, key_needs_update and key_ready are low and a new key can be loaded into the pre-processor at any time. After the new key is loaded using the SDI port, key_ready goes high. After the instruction ACTIVATE_KEY is received at the PDI port, the key_needs_update goes high. Please note that the above two events can occur in an arbitrary order. After key_ready and key_needs_update are both high, and the cipher core is either in the period between reset and the first input, or in the period between two consecutive inputs, the cipher core should read the new key. After the key is read, key_updated signal should be set to high. The key_updated signal should be deactivated at the end of processing of the current input. If a user wants to use the same key for the subsequent input data, ACTIVATE_KEY instruction can be omitted from the PDI input port. In this case, the processing of new data will start as soon as an instruction describing the way of processing a new input is decoded (which is indicated by bdi_proc set to high).

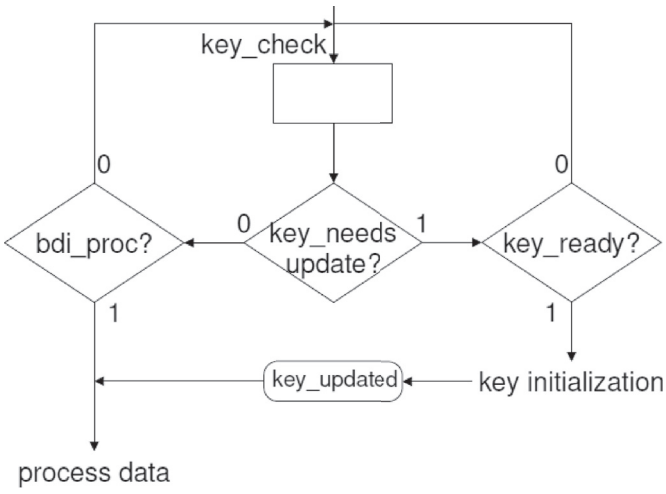

Fig. 20. A part of the Algorithmic State Machine (ASM) chart describing a way in which the cipher core controller may handle key loading and key activation.

In summary, the cipher core should monitor the key_needs_update port prior to processing any new input. If key_needs_update is high, the cipher core should wait for key_ready=1, and then read the new key, and acknowledge its receipt using the key_updated output. If key_needs_update is low and the first instruction describing the way of processing a new input is decoded (bdi_proc=1), then the cipher core should move directly to processing a new input using a previous key. If none of these two events is detected, the cipher core should remain in the same state. The described behavior is shown in Fig. 20.

The key initialization and process data are two separate states that operate depending on the requirements of a specific cipher. In the second group of states, Process associated data, the core continuously waits for the next AD block until the bdi_eot signal becomes active. This signal indicates that the current block is the last block of associated data. The state machine needs then to process this last block, and proceed to the next group of states, responsible for encryption and decryption of data. If the first block read by the cipher core is not of type AD (bdi_ad=0), then associated data is assumed to be empty. If the last block of AD (bdi_ad=1 and bdi_eot=1) is also the last block of input (bdi_eoi=1), then the message/ciphertext is assumed to be empty.

The third group of states, process message/ciphertext, should operate in the similar way as the second group, and should similarly progress to the next group of states when the last block of message or ciphertext is processed. In this group of states, bdi_ad should remain inactive for each input block to indicate that the current block is not an associated data block. A corresponding output data block should be passed to the post-processor using the bdo port with an accompanied active bdo_write control signal. After each block of associated data, message, or ciphertext is read by cipher core, the bdi_read output must be activated for one full clock cycle. This action clears control inputs, such as bdi_eot and bdi_eoi that may need to be checked at a later time.

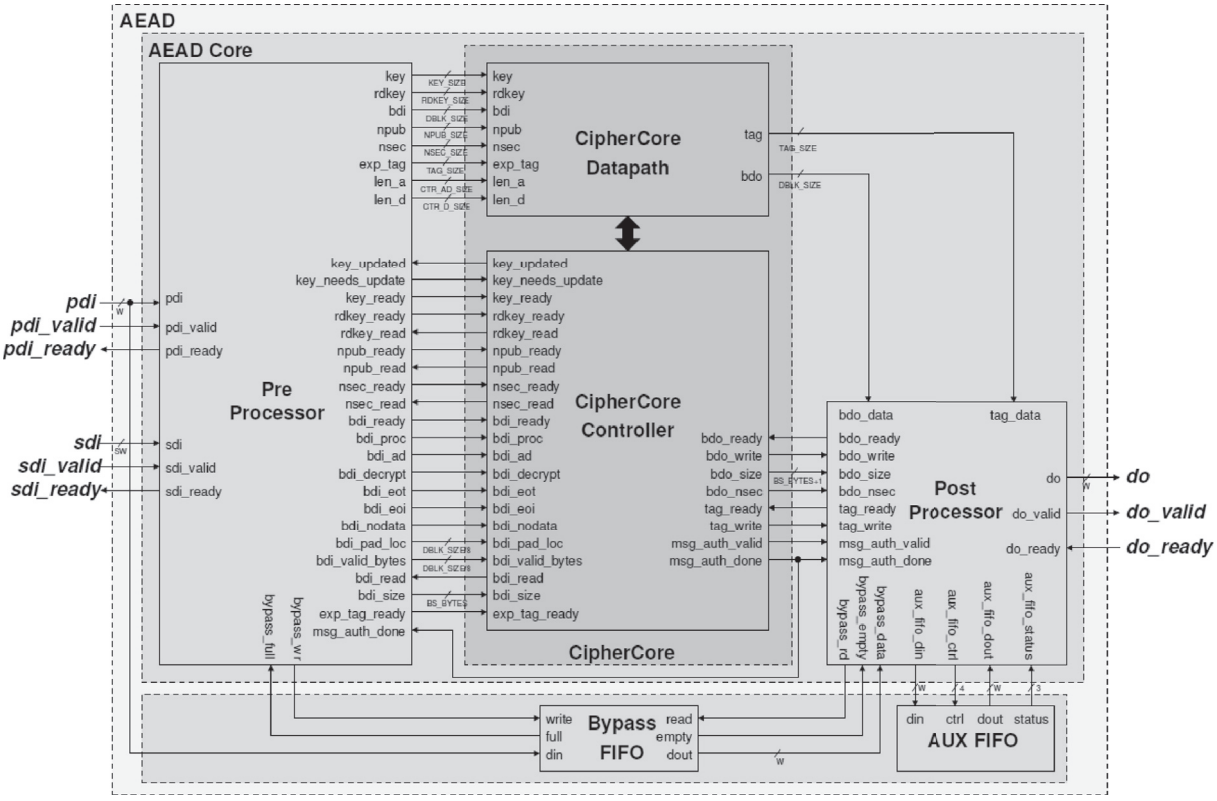

Fig. 21. High-level block diagram of the AEAD core.

At the same time, this action cannot be delayed because doing so would stall the pre-processor and prevent it from loading any subsequent data block using the PDI input. As a result, bdi_eot and bdi_eoi must be registered at the latest in the clock cycle when the acknowledgment signal bdi_read is generated. Only registered values of these inputs should be checked at a later time.

In the last group of states, Generate/verify an authentication tag, during the authenticated encryption, the core should generate a new tag and pass it to the post-processor, using ports tag and tag_write. During the authenticated decryption, msg_auth_done should be activated, and the msg_auth_valid port should be used to output the result of authentication.

A block diagram illustrating all the necessary signals for the AEAD core consisting of: cipher core, pre-processor and post-processor is shown in Fig. 21.

5.2. Design flow

The DPR flow is carried out using Xilinx Vivado tool. A complete design for the encryption module is shown in Fig. 22. A similar design is tailored for the decryption module with the same components. The system is controlled by the ARM Cortex A9 microprocessor on the Processing System (PS) side. The processor communicates with the Programmable Logic (PL) side through the AXI bus interface. The static part consists of: the PR controller, the ICAP primitive, the AXI connections, AEAD top, FIFOs, UART in addition to the LFSR in the encryption side. The dynamic part contains Reconfigurable Partition (RP) that holds the 5 Reconfigurable Modules (RMs) corresponding to the 5 ciphers. The partial bitstreams for these RMs are stored in the SD card to be loaded through a software code that interfaces the PS with the Personal Computer (PC) via UART connection. The operating frequency of PL is 10 MHz while the PS operates at a frequency of 667.66 MHz.

A complete flow of a full encryption/decryption operation is described as follows:

i. The sender starts the encryption module by entering the seed into the LFSR.
ii. The LFSR generates a pseudo-random sequence, the sequence's least three significant bits are used as the ID to specify the cipher to be used.
iii. The AEAD top switches to the desired cipher and loads the test vectors into the FIFOs from the text files.
iv. The output data is stored into the output FIFO until the encryption is done.
v. When done, the UART transmitter sends the enable signal along with the ID of the cipher to the decryption side.
vi. The decryption module switches to the corresponding cipher and starts receiving the data encrypted through the UART.
vii. When the decryption is done, a status message is sent back to the encryption module through the UART to mark the success of the full operation and allow for the next hop.

6. Cryptanalysis and security vulnerabilities

In this section, detailed crypt-analysis of the security algorithms FPGA implementations and specific hardware solutions to reduce the danger of some attacks for each individual algorithm are discussed.

6.1. ASCON

Pure ASCON is vulnerable to side channel attacks as shown in Ref. [24] and on average, only 500 power traces can reveal the correct key.

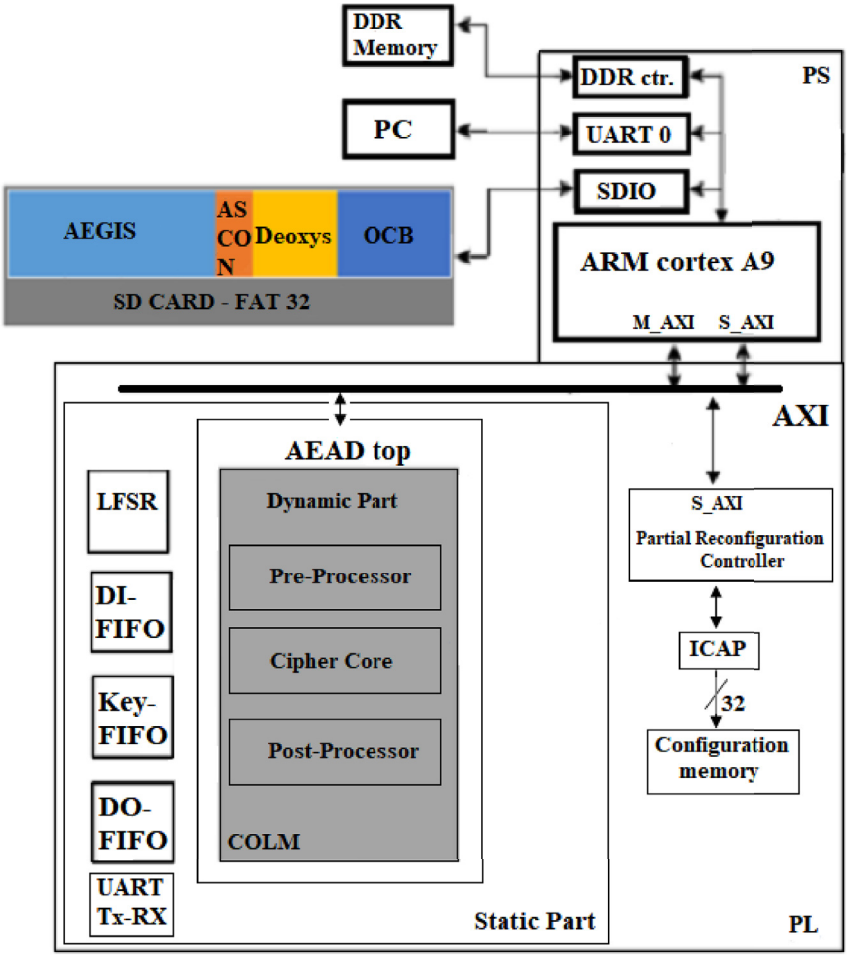

Fig. 22. Hardware design of the encryption module.

This is due to the weakness of using 5-bit S-Boxes. While it is the non linear part of the algorithm, its input can be easily guessed by generating the power traces of all possible input combinations (2^5 possibilities) and then correlate each of them with the true power trace obtained from the system under attack. Such procedure is done at either the first or last round of the S-Box to be near known input (plain-text) or output (cipher-text) respectively. By repeating this procedure against all vulnerable S-Boxes the whole secret key can be revealed.

Deployed solution here is inspired from KASUMI side channel attack countermeasure proposed in Ref. [25] by masking the power consumption information in the S-Box with another random S-Box denoted by S∗, As shown in Fig. 23. While the original S-Box is accepting the 5-bit input X_i, the masking S∗-Box is accepting another random 5-bit input X_{*i} and the result is then swapped twice, This design is evaluated in Ref. [25] and totally masks the power traces and correlation trace of the correct key doesn't produce a correlation peak at the time instant corresponding to key processing.

The ASCON 5×5 S-Box is implemented as bit-slicing logic functions not as look-up tables as shown in Fig. 24 to overcome cache timing attacks.

6.2. AEGIS

The most obvious weakness is the chosen plain-text attacks, to ensure its avoidance, the same initialization vector mustn't be reused with the same key, this is ensured in our design through the use of 128-bit LFSR to generate random IVs agreed upon on both sides.

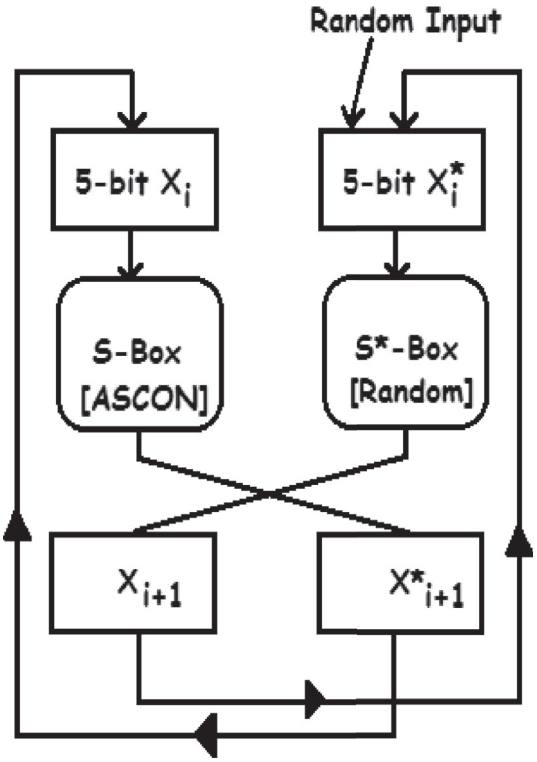

Fig. 23. The Masking S∗-Box to countermeasure the SCA.

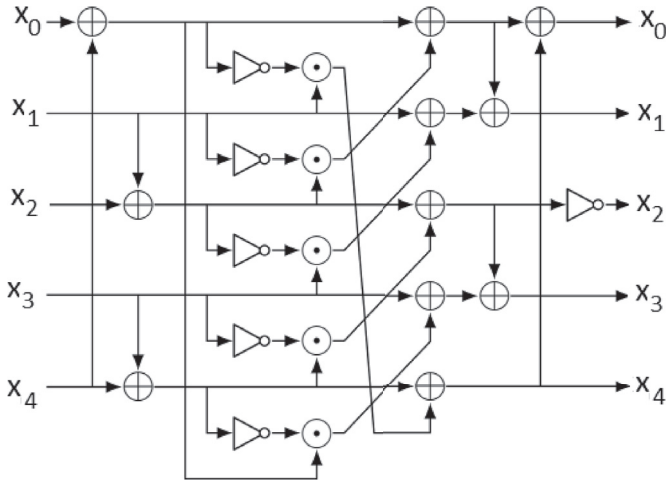

Fig. 24. The Bit-Slicing implementation of the ASCON 5 × 5 S-Box [8].

The used LFSR is tapped at the following locations: 128, 126, 101,99 which according to Ref. [26] generates the maximum length possible for 128-bit LFSR.

Differential attack against the initialization is more time-wise expensive than the brute force as there are 50-AES round functions in the initialization.

Weak states are of equal 16-bytes in the AEGIS state, these weak states count = 2^{128}, noting that only 5 states are related to the input key out of the AEGIS eight states, so the total number of states = $2^{5*128} = 2^{640}$ states, so these weak states probability = $2^{128-640} = 2^{-512}$ which is very tiny probability but even this tiny possibility is avoided in the hardware through by discarding IVs resulted from the 128-bit LFSR responsible for controlling the IV values generation that participates in all-equal states.

6.3. COLM version 1

Reusable re-forgery attacks on both COLM and OCB were described in Ref. [27]. Reforge-ability describes the complexity to find subsequent forgeries if a forgery is found.

6.4. Deoxys version 1.3

It is resistant against single-key differential attacks, which is achieved simply through the diffusion layer of S-Box used inside AES deployed by the deoxy algorithm. Also resistant against the related-key differential attacks which needs impractical complexity to reveal the key as shown in Ref. [28] the attack needs a time complexity of $2^{173.1}$, 10-round encryptions and a data complexity of 2^{135} plaintexts.

6.5. OCB

The OCB algorithm is timing-attack resistant and therefore is slow comparing to other algorithms, the proof showing mathematically how OCB is timing attack resistant is shown in Ref. [29].

Hopping between the five algorithms using dynamic partial reconfiguration saves the utilization area and combines the different security advantages over the discussed vulnerabilities efficiently.

7. Implementation results

7.1. Resources utilization

As shown in Table 1, the resource utilization of each configuration is calculated for a Xilinx XC7Z020LG484-1 Zynq FPGA [30] after synthesis using Xilinx Vivado 2015.2. The five configurations correspond to the five ciphers: AEGIS, ASCON, COLM, Deoxys and OCB respectively. It is important to note that these results are based on the original implementations by the designers of the ciphers. There are no modifications applied to optimize for lower resources utilization. The main purpose is to prove the flexibility of the proposed design to embrace any configuration.

Table 2 presents the resources utilized by the static components of the design including the inputs/outputs FIFOs, UART transmitter/receiver and the LFSR. PDI FIFOs exhibits the largest area with 801 slice LUTs, 277 slice registers and no F7 or F8 MUXs (Fx MUX is equivalent to a LUT that can implement an x input function). This is because these FIFOs have largest input/output port width of 256 bits and a depth of 32 registers. UART Rx consumes a significant amount of slice LUTs compared to UART Tx as it handles potential metastability resulting from different clock domains by double registering the input data.

By integrating the utilization results for the dynamic and static parts as shown in Fig. 25, it is found that DPR helps in decreasing the utilization by 57% when compared to the traditional design with the five configurations physically mounted onto the FPGA at the same time without switching. These results apply to both encryption and decryption modules as they are using the same components except for the LFSR which is amortized over the whole design area.

Table 1
Resources utilization of the dynamic configurations.

Configuration	Slice LUTs	Slice registers	F7 MUXs	F8 MUXs	BRAM tiles
AEGIS	7250	2117	2052	1024	0
ASCON	1321	890	2	0	0
COLM	7547	2686	1137	376	4
Deoxys	2971	1593	641	256	0
OCB	4009	1612	593	200	4

Table 2
Resource utilization of the static modules.

Component	Slice LUTs	Slice registers	F7 MUXs	F8 MUXs	BRAM tiles
LFSR	4	3	0	0	0
DO/PDI FIFOs	801	277	0	0	0
SDI FIFO	126	45	0	0	0

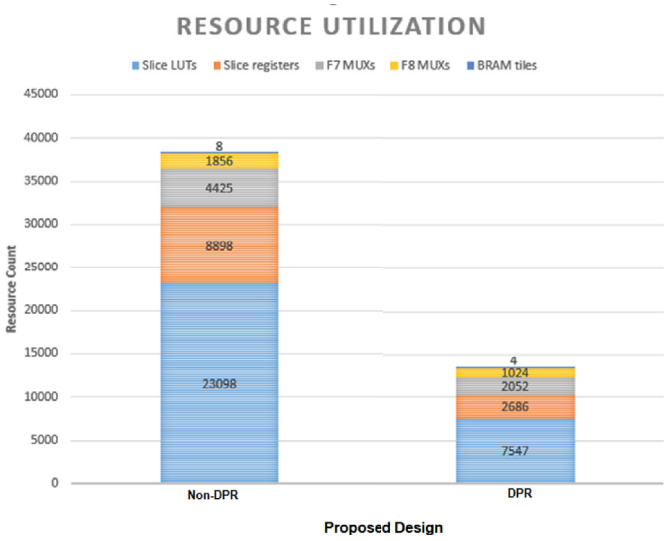

Fig. 25. Comparison between Non-DPR and using DPR implementation.

Table 3
Dynamic power consumption.

Configuration	Dynamic Power Consumption (mW)
AEGIS	9.26
ASCON	2
COLM	10.74
Deoxys	6.3
OCB	4.14
DPR controller	9.2

7.2. Power consumption

In Table 3, dynamic power consumption is calculated for each configuration for both encryption and decryption modules at an operating frequency of 10 MHz using Xilinx Vivado 2015.2 power analysis tool in addition to the power consumed by the AXI-HWICAP DPR controller. Power calculations are based on 3 input parameters to the analysis tool: design source clock frequency, I/O data ports and internal control signals nodes switching activity. Source clock frequency is predetermined though a user constraint file while switching activity of all internal nodes is provided through a SAIF extension file that is generated through timing simulation of the design at worst case scenarios. It is important to note that the power consumed by the DPR controller is independent of that of used clock frequency for the PL, as it is dependent on the bandwidth of internal configuration access port which is configured at a maximum value of 667 MHz. With an average consumption of 6.48 mW, the proposed design reduces power consumption by 80% when compared to a conventional design using the five configurations in parallel without exploiting the capabilities of DPR, taken into consideration that no optimizations were performed on the used encryption algorithm cores as the purpose was to prove the ability of the proposed design to embrace as many configurations as needed.

7.3. Reconfiguration time

The speed of configuration is directly related to the size of the partial bit file and the bandwidth of the configuration port. The reconfiguration time can be calculated by dividing the size of the bitstream (in bits) by the throughput of the ICAP [31]. The calculated configuration time is 1.67 ms for the five configurations as they are using the same RP with fixed area on the FPGA. Although the reconfiguration time is the main drawback of DPR, it is compensated by allowing the ARM processor to do other tasks during this time which helps in increasing the speed of any reconfigurable design. Moreover, the effect of the reconfiguration time on the total time required for data transmission is highly dependent on the frequency at which the PL operates as it directly affects the latency of the encryption/decryption modules, hence, the time it takes to transfer each bit per encryption/decryption message. It is also important to take into consideration the technology that could be used to transfer the encryption/decryption data between the 2 modules as it controls the maximum distance covered by the data to travel across 2 nodes and accordingly the time taken through transmission.

Table 4
Throughput of the five configurations.

Configuration	Throughput (Mbps)
AEGIS	51.2
ASCON	30.5
COLM	45.7
Deoxys	27.2
OCB	49.2

Table 5
Comparison between the proposed design and [32].

Parameter	Proposed Design	[32]
device	XC7Z010clq225-3	XC7Z010clq225-3
power [mW]	72.6	170
Throughput [Mbps]	510	1280
Efficiency [Mbps/slice]	0.54	2.45
confidentiality	yes	yes
Integrity	yes	No
Authentication	yes	No
second dimension of security	yes	No

7.4. Throughput

Table 4 presents the throughput of each cipher used in the design at a frequency of 10 MHz. The average throughput of the design is estimated under the assumption that each cipher is used to send one encryption/decryption message and was found to be 40.8 Mbps. Moreover, the maximum achievable throughput is limited by the minimum of either the bandwidth of ICAP or the maximum frequency of one of the used configurable ciphers.

8. Comparison

Table 5 shows comparison between the proposed design and [32] which is using AES as an encryption algorithm for an IoT security system, when the two designs run at 100 MHz. It shows that the average power consumption for the proposed design is only 72.6 mW and it supports three aspects of the cryptography goals which are: confidentiality, integrity and authentication in addition to the second dimension of security whereas the study [32] supports only confidentiality and the power consumption for it is 170 mW. Although other parameters are in favor of [32], it is notable to mention that it is possible to enhance the proposed design results by either optimizing the used algorithms area or by using algorithms which utilize less resources as area utilization is of much higher priority than throughput calculations when it comes to the field of IoT.

9. Conclusion

In this paper, a new design that adds a new dimension of security for constrained IoT devices using the concept of algorithm hopping, in analogy to the well known frequency hopping technique, is introduced. DPR technology is used for switching between five lightweight ciphers that are currently under review in the CAESAR contest and provides a reduction of 58% and 80% in area utilization and power consumption respectively. The design is mounted on a Xilinx XC7Z020LG484-1 Zynq FPGA and synthesized using Xilinx Vivado 2015.2.

Acknowledgement

This work was supported by the Egyptian Information Technology Industry Development Agency (ITIDA) under ITAC Program PRP2018.R25.23.

References

[1] R. Minerva, A. Biru, D. Rotondi, Towards a Definition of the Internet of Things (IoT), IEEE, May 2015 (May 2015).
[2] C. Bekara, Security issues and challenges for the IoT-based smart grid, Procedia Comput. Sci. 34 (2014) 532–537.
[3] M. Katagi, S. Moriai, Lightweight Cryptography for the Internet of Things, Sony Corporation, 2008, pp. 7–10.
[4] N. Samir, Y. Gamal, A.N. El-Zeiny, O. Mahmoud, A. Shawky, A. Saeed, H. Mostafa, Energy-adaptive lightweight hardware security module using partial dynamic reconfiguration for energy limited internet of things applications, in: IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 2019https://doi.org/10.1109/ISCAS.2019.8702315.
[5] S. Koteshwara, A. Das, Comparative study of Authenticated Encryption targeting lightweight IoT applications, IEEE Des. Test 34 (2017) 26–33, https://doi.org/10.1109/MDAT.2017.2682234.
[6] Cryptographic Competition, 2014 (2014), https://competitions.cr.yp.to/caesar-call.html.
[7] H. Wu, B. Preneel, AEGIS: a fast authenticated encryption algorithm, in: T. Lange, K. Lauter, P. Lisonek (Eds.), Selected Areas in Cryptography SAC 2013. Lecture Notes in Computer Science, vol. 8282, Springer, Berlin,Heidelberg, 2014.
[8] C. Dobraunig, M. Eichlseder, F. Mendel, M. Schlffer, Ascon v1.2, Submission to the CAESAR Competition, 2016, http://competitions.cr.yp.to/round3/asconv12.pdf, http://ascon.iaik.tugraz.at.
[9] J. Daemen, Permutation-based Encryption, Authentication and Authenticated Encryption, DIAC- Directions in Authenticated Ciphers, July 2012 (July 2012).
[10] E. Andreeva, A. Bogdanov, N. Datta, A. Luykx, B. Mennink, M. Nandi, E. Tischhauser, K. Yasuda, COLM V1, Submission to CAESAR Competition, 2015.
[11] C. Cid, T. Huang, T. Peyrin, Y. Sasaki, L. Song, A security analysis of deoxys and its internal tweakable block ciphers, IACR Trans. Symmetric Cryptol. 3 (2017) 73–107.
[12] T. Krovetz, P. Rogaway, The OCB Authenticated-Encryption Algorithm, RFC 7253, May 2014https://doi.org/10.17487/RFC7253 (May 2014).
[13] N.H. Motlagh, Frequency hopping Spread Spectrum: an effective way to improve wireless communication performance, Adv. Trends Wireless Commun. 2011 (2011).
[14] Xilinx Inc, Efficient Shift Registers, LFSR Counters, and Long PseudoRandom Sequence Generators (Jul. 1996), Jul. 1996.
[15] C. Maxfield, Book: the Design Warrior's Guide to Fpgas, Newnes, 2004 (2004).
[16] M.A. Bahnasawi, K. Ibrahim, A. Mohamed, M.K. Mohamed, A. Moustafa, K. Abdelmonem, Y. Ismail, H. Mostafa, ASIC-oriented comparative review of hardware security algorithms for internet of Things applications, in: IEEE International Conference on Microelectronics (ICM 2016), 2016, pp. 285–288, https://doi.org/10.1109/ICM.2016.7847871.
[17] A. Hassan, R. Ahmed, H. Mostafa, H.A.H. Fahmy, A. Hussien, Performance evaluation of dynamic partial reconfiguration techniques for software defined radio implementation on FPGA, in: IEEE International Conference on Electronics, Circuits, and Systems (ICECS), 2015https://doi.org/10.1109/ICECS.2015.7440279.
[18] Xilinx Inc, Vivado Design Suite Partial Reconfiguration User Guide UG909 (Apr. 2017), Apr. 2017.
[19] K. Vipin, S.A. Fahmy, ZyCAP: efficient partial reconfiguration management on the xilinx Zynq, IEEE Embed. Syst. Lett. 6 (3) (2014) 41–44.
[20] A. Kamaleldin, A. Mohamed, A. Nagy, Y. Gamal, A. Shalash, H. Mostafa, Design guidelines for the high-speed dynamic partial reconfiguration based software defined radio implementations on xilinx Zynq FPGA, in: IEEE International Symposium on Circuits and Systems (ISCAS), May 2017, pp. 1–4, https://doi.org/10.1109/ISCAS.2017.8050456.
[21] Xilinx Inc, AXI HWICAP PG134 (Oct. 2016), Oct. 2016.
[22] K. Khatib, M. Ahmed, A. Kamaleldin, M. Abdelghany, H. Mostafa, Dynamically reconfigurable power efficient security for internet of Things devices, in: 7th International Conference on Modern Circuits and Systems Technologies (MOCAST), 2018https://doi.org/10.1109/MOCAST.2018.8376645.
[23] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, M.U. Sharif, K. Gaj, GMU hardware API for authenticated ciphers, in: International Conference on Reconfigurable Computing and FPGAs, Dec. 2015, pp. 1–8.
[24] C.D.C.E. Hannes Gross, Erich Wenger, Ascon hardware implementations and side-channel evaluation, in: International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE) (2018), 2016https://doi.org/10.1016/j.micpro.2016.10.006.
[25] S.T. Devansh Gupta, B. Mazumdar, Correlation power analysis on KASUMI: attack and countermeasure, in: Microprocessors and Microsystems (MICROPROCESS MICROSY), Elsevier, 2018, pp. 142–156, https://doi.org/10.1007/978-3-030-05072-6_9(2016).
[26] xilinx.com (Linear Feedback Shift Register Taps) , https://www.xilinx.com/support/documentation/application_notes/xapp052.pdf.
[27] S.L. Christian Forler, Eik List, J. Wenzel, Reforgeability of Authenticated Encryption Schemes, 2017.
[28] X.D. Rui Zong, X. Wang, Related-tweakey Impossible Differential Attack on Reduced-Round Deoxys-Bc-256, 2019.
[29] T. Krovetz, P. Rogaway, The Software Performance of Authenticated-Encryption Modes, International Association for Cryptologic Research, 2011.
[30] Xilinx Inc, ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC UG850 (Sep. 2015), Sep. 2015.
[31] R.L. Roux, G.V. Schoory, P.V. Vuuren, Block RAM-based architecture for real-time reconfiguration using Xilinx FPGAs, S. Afr. Comput. J. 2015 (2015).
[32] S.M. Soliman, B. Magdy, M.A.A. El-Ghany, Efficient implementation of the AES algorithm for security applications, in: 29th IEEE International System-On-Chip Conference (SOCC), Seattle, WA, 2016, pp. 206–210, https://doi.org/10.1109/SOCC.2016.7905466.