Journal of Circuits, Systems, and Computers
Vol. 25, No. 7 (2016) 1650078 (19 pages)

DOI: 10.1142/S021812661650078X

Efficient Hardware Implementation of Ultralightweight RFID Mutual Authentication Protocol*

Umar Mujahid†, Atif Raza Jafri‡ and M. Najam-ul-Islam§

Department of Electrical Engineering,
Bahria University, Islamabad Campus,
ICT Capital, Pakistan
†*umar.mujahid@bui.edu.pk*
‡*atif.raza@bui.edu.pk*
§*najam@bui.edu.pk*

Received 6 September 2015
Accepted 18 January 2016
Published 17 March 2016

Security and privacy are the two major concerns of radio-frequency identification (RFID) based identification systems. Several researchers have proposed ultralightweight mutual authentication protocols (UMAPs) to ensure the security of the low cost RFID tags in recent years. However, almost all of the previously proposed protocols have some serious security flaws and are vulnerable to various security attacks (full disclosure attack, desynchronization attack, impersonation attack, etc.). Recently, a more sophisticated and robust UMAP: Robust confidentiality integrity and authentication (RCIA)[1] [U. Mujahid, M. Najam-ul-Islam and M. Ali Shami, RCIA: A new ultralightweight RFID authentication protocol using recursive hash, *Int. J. Distrib. Sens. Netw.* **2015** (2015) 642180] has been proposed. A new ultralightweight primitive, "recursive hash" has been used extensively in the protocol design which provides hamming weight unpredictability and irreversibility to ensure optimal security. In addition to security and privacy, small chip area is another design constraint which is mandatory requirement for a protocol to be considered as ultralightweight authentication protocol. Keeping in view the scenario presented above, this paper presents the efficient hardware implementation of the RCIA for EPC-C1G2 tags. Both the FPGA and ASIC implementation flows have been adopted. The FPGA design flow is primarily used to validate the functionality of the proposed hardware design whereas ASIC design (using TSMC 0.35 μm library) is used to validate the gate count. To the best of our knowledge, this is the first FPGA and ASIC implementation of any ultralightweight RFID authentication protocol. The simulation and synthesis results of the proposed optimal hardware architecture show the compatibility of the RCIA with extremely low cost RFID tags.

Keywords: RFID; UMAP; FPGA; ASIC; security.

*This paper was recommended by Regional Editor Piero Malcovati.
†Corresponding author.

1. Introduction

Radio-frequency identification (RFID) is one of the most growing identification schemes in the field of ubiquitous computing. Non-line-of-sight capability of the RFID systems massively increases its deployment compared to other contended identification schemes. An RFID system is mainly composed of three components: tag, reader and back-end database. Usually, an RFID tag consists of two main parts: (i) integrated circuit (IC) (memory registers and logic gates) for data storage and other computational operations, (ii) antenna for information transmission and reception. A commercial RFID tag contains Electronic Product Code (EPC), which represents its unique and static identity (ID). The tag stores this EPC and other protocol specific relevant information in its memory and responds with this data on reader's query. Upon receiving of the tag's information, the reader forwards this data towards back-end database and then back-end database checks the legitimacy of tag's ID.

RFID standardization is a major issue in fortifying the massive investment for development and swift deployment. There are the two standardization regulations: ISO/IEC14443 A/B (Ref. 33) and Electronic Product Code Global Class-1 Generation-2 (EPC-C1G2).[28] ISO standard provides standardized framework for development of passively powered but mostly high cost RFID tags. EPC-C1G2 provides the framework for development of low cost and extremely small size RFID tags. The demand for low cost tags (0.05–0.1$) limits us to use simple logical operations in authentication protocols' development. Typically, such tags can store 32–1 K bits and can support 250–4 K logic gates for security related tasks. Table 1 summarizes the attributes of EPC-C1G2 based RFID tags.

As far as security protocols are concerned, Chien[3] categorized the security (authentication) protocols into four main classes on the basis of hardware constraints: full-fledged, simple, lightweight and ultralightweight.

(i) Full-fledged authentication protocols: This class refers to those protocols which can incorporate traditional cryptographic functions and support one-way hash functions, public key algorithms or symmetric encryption techniques, etc.

Table 1. Properties of EPC-C1G2 low cost RFID tags.

S. No.	Performance metrics	Attributes
1.	Standard	EPC-C1G2
2.	Memory storage	32–1 K bits
3.	Overall logic gate equivalent (GE)	5–10 K
4.	Logic GE for security related tasks	250–4 K
5.	Power source (active/passive)	Passive
6.	Price	0.05–0.1$
7.	Resistance to passive attacks	Yes
8.	Resistance to active attacks	No (but depends upon the protocol)
9.	Resistance to physical attacks	No

(ii) Simple authentication protocols: This class supports the pseudorandom number generators (PRNGs) and one-way hash function on the tag's chip.
(iii) Lightweight authentication protocols: This class refers to those protocols which may support only lightweight random number generators and cyclic redundancy check (CRC) type simple functions.
(iv) Ultralightweight authentication protocols: These protocols involve only simple logical operations such as bitwise XOR, AND, OR, etc.

Ultralightweight mutual authentication protocols (UMAPs) provide extremely low security. This is mainly due to the wide use of simple T-functions[26] for designing the security algorithms, in addition to traditional cryptographic functions (which are resource hungry). However, inclusion of non-triangular operations (rotation, permutation, recursive hash, etc.) in UMAPs design augments the resistance against various types of security attacks. A brief overview of numerous ultralightweight authentication protocols and their security attacks is described below:

1.1. *Overview of UMAP family protocols*

In 2006, Periz-Lopez *et al.*[4–6] introduced the concept of lightweight cryptography for RFID systems. They laid down the first stepping stone of ultralightweight cryptography and proposed three UMAPs for the low cost RFID tags: lightweight mutual authentication protocol (LMAP), efficient-lightweight mutual authentication protocol (EMAP) and minimalist mutual authentication protocol (M2AP). In all the three UMAPs, each tag stores a static ID, an index pseudonym IDS and four internal keys (K_1, K_2, K_3, K_4) of n bits. Because of limited computational capabilities, the tag only performs simple bitwise logical operations (XOR, AND, OR, Modulo-2 addition, etc.). Since the computation of pseudorandom numbers is a resources demanding operation, it will be performed by the reader. These protocols mainly involve three steps: tag identification, mutual authentication and pseudonyms and keys updating. The authors estimated that both LMAP and M2AP require 1 K GEs approximately, while EMAP requires only 468 GEs (for 96-bit variable length) which efficiently fulfills the requirements of EPC-C1G2 tags. However, Alomair *et al.*[7] and Barasz *et al.*[8,9] found some serious vulnerabilities in the protocols and proposed passive attacks to reveal the concealed secret ID. They directly exploited the inherent poor diffusion properties of T-functions and hence retrieved all the concealed secrets with 100% success rate.

In 2007, Chien[3] formally named such low cost security solutions as "ultralightweight cryptography" and proposed more sophisticated ultralightweight protocol: strong authentication and strong integrity (SASI). The basic protocol structure was similar to its preceding protocols; however, a new non-triangular function "rotation" was introduced to combat against various desynchronization and full disclosure attacks. Later, D'Arco and De Santis,[10] Sun *et al.*[11] and Avoine

et al.[12,13] highlighted many loopholes and vulnerabilities of the SASI protocol. They proposed desynchronization and full disclosure attacks on the SASI protocol. The former attack breaks the synchronization between the reader and the tag while the latter attack discloses the unique secret concealed ID of the tag. These attacks place the SASI protocol among vulnerable UMAPs.

In 2008, Peris-Lopez *et al.*[14] improved their previous work and presented another ultralightweight authentication protocol: Gossamer. A new ultralightweight primitive "MixBits" has been used in Gossamer to avoid previous loopholes which occur due to extensive use of simple T-functions in protocol designs (such as in LMAP, EMAP and M2AP). Although the MixBits function provides strong resistance against various full disclosure attacks, Yeh and Lo,[16] Tagra *et al.*[15] and Bilal *et al.*[17] highlighted the poor design of the protocol and proposed some active desynchronization and denial-of-service (DoS) attacks on the protocol. In 2012, Zubair *et al.*[18] assimilated a message counter with Gossamer protocol to avoid multiple DoS attacks. The desynchronization issue still makes Gossamer practically infeasible for the low cost RFID systems.

To avoid impersonation and desynchronization attacks, David and Prasad[19] introduced a concept of "day certificate" for the reader in 2009. But the David–Prasad protocol was also found to be vulnerable against many simple security attacks as it also involves simple T-functions in protocol messages. In 2010, Hernandez-Castro[20] proposed traceability and full disclosure attack (Tango) on the protocol. The Tango attack first selects the good approximations (GAs) for secrets and then combines and compares these GAs to reveal the unique ID. Barrero *et al.*[21] improved the Tango attack and used genetic programming to resolve the issue of exhaustive searching of optimal GAs. Most of the security attacks proposed for UMAPs are *ad hoc* (protocol specific) and cannot be applied to a broader class of the protocols. The Tango attack was the first formal security analysis framework which can be applied to broader class of UMAPs for security analysis.

In 2012, Tian *et al.*[2] presented quite an interesting UMAP (RAPP) using bitwise "permutation". However in permutation operation, the hamming weight (wt) of the input and output variables ($w(\mathrm{Per}(X, Y) = \mathrm{wt}(X))$ remain the same. Bagheri *et al.* exploited this weakness and highlighted the traceability attack on RAPP.[34] In 2013, Ahmadian *et al.*[22] identified a desynchronization attack on the RAPP with optimal success probability (0.25). Wang *et al.*[23] also used the same property of the permutation operation and proposed a powerful full disclosure attack on the RAPP protocol.

Most of the previously proposed UMAPs have similar pitfalls in their designs such as the use of T-functions, linear functions (Ro, Per, etc.) and poor messages composition, so on, which lead towards their full disclosure. Recently, a new UMAP has been proposed to provide robust confidentiality, integrity and authentication (RCIA).[1] RCIA introduces a more sophisticated nonlinear function (recursive hash) which enhances the diffusion properties of the protocol messages optimally. Multiple

security analysis of the RCIA protocol shows that it can withstand against all known formal and *ad hoc* attacks. The detailed description of RCIA is presented in Sec. 2.

1.2. *Contribution and organization*

In this paper, we present the efficient hardware implementation of the RCIA protocol for EPC-C1G2 tags (low cost tags) using both FPGA and ASIC level design flows. For FPGA based prototyping, we have used ModelSim to validate the design and Xilinx ISE 12.3 Design Suite has been used for circuit synthesis and resource approximations. We explore the fully occupied look-up tables (LUTs), slice registers and fully used LUTs/FFs of FPGAs (Spartan-6 and Virtex-5). For ASIC, we have used Leonardo Spectrum (TSMC 0.35 μm) to approximate the number of gates. To the best of our knowledge, this is the first paper that reports an implementation of UMAP for both FPGA and ASIC design approaches. Martin *et al.*[25] and Huang *et al.*[24] also reported hardware implementations of the similar types of authentication protocols using ASIC and FPGA designs, respectively, however, they have targeted the lightweight class protocols. There is a variety of the protocols with diverse primitives, so this implementation will act as the stepping stone for further development of the reconfigurable ultralightweight cryptographic processors (presented in Sec. 5).

The rest of the paper is organized as follows. In Sec. 2, newly proposed ultralightweight RFID authentication protocol (RCIA) has been presented. Design and hardware architecture of the RCIA is discussed in Sec. 3. Section 4 describes the circuit synthesis and simulation results which are followed by the reconfigurable architecture design for the UMAPs in Sec. 5. Finally, conclusion and future recommendations are discussed in Sec. 6.

2. Overview of Ultralightweight RFID Authentication Protocol: RCIA

For better understanding and completeness, in this section, we provide an overview of ultralightweight RFID authentication protocol: RCIA. In RCIA, tags mainly use three bitwise logical operations: AND, XOR and left rotation (Rot). A new ultralightweight primitive, "recursive hash" (R_h), has also been used in protocol messages, which provides hamming weight unpredictability and irreversibility to avoid the multiple attacks. Recursive hash of any variable is a multiple step operation which can be computed as follows:

(A) Computation of recursive hash (R_h) function:

Suppose let 'X' is a k-bit string, where

$$X = x_{k-1}x_{k-2}\ldots x_0, \quad x_i \in \{0,1\}, \quad i = 0,1,\ldots,k-1\,. \tag{1}$$

Then the computation of recursive hash of X, $R_h(X)$, involves following steps:

(i) Decimate the string X into 'S' number of memory blocks, having equal number of bits 'b' in each memory block ($S = k/b$).
(ii) After extraction of random numbers (n_1, n_2) from messages (A, B in the case of RCIA), tag uses the following equation to compute the Seed (index of memory block) for recursive hash:

$$\text{Seed} = \text{hw}(n_1 \oplus n_2) \mod S\,, \tag{2}$$

where hw represents the hamming weight.
(iii) Calculated Seed will select the corresponding memory block (S_i) of the decimated string X and then the tag will perform three steps to compute the final recursive hash, $R_h(X)$:
   (a) Take XOR between the selected memory block (S_i) and other memory blocks except the block (S_i) itself.
   (b) Left rotate block (S_i) with itself; $\text{Rot}(S_i, \text{hw}(S_i))$.
   (c) Concatenate the resultant rotated memory block with XORed memory blocks (defined index location).

To better understand the concept of recursive hash (R_h) function, consider the following example:

Example 1. Assume (for 32-bit architecture)

$$X = 11010010100011001001110100111010\,,$$
$$n = 32, \quad b = 8, \quad S = S_3 \ldots S_0 \quad \text{and} \quad \text{Seed} = 1\,.$$

Then the recursive hash of X will be

$$R_h(X) = 01001111000100011011001110100111\,.$$

Figure 1 shows the stepwise computation of Example 1.
$\text{Rot}(X, Y)$ is basically cyclic left rotation of X according to $\text{hw}(Y)$.[11]

(B) Description of RCIA Protocol:

RCIA mainly involves three phases: pre-identification, mutual authentication and pseudonym updating. In pre-identification stage, reader sends "Hello" message to the tag. The tag responds with its current IDS. If the reader finds the same IDS in its database then it proceeds to next stage (mutual authentication phase) otherwise the reader sends an error message (another "Hello" message) towards the tag.

Upon receiving of error message, the tag responds with its IDS_{old} (previous IDS value). This time legitimate reader will definitely find the matched entry in the database and proceeds to the next step.

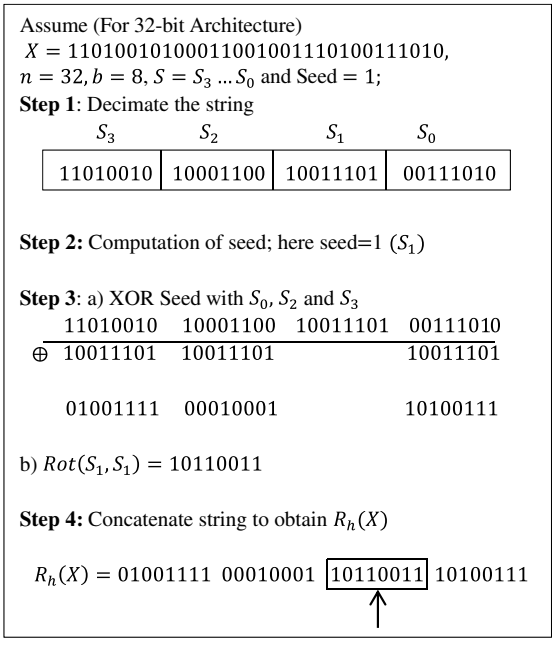

Fig. 1. An example for computation of recursive hash.

In the mutual authentication phase, reader computes two random numbers (n_1, n_2), conceals them in messages (A, B) and then sends message burst $A\|B\|C$ to the tag. The messages A, B and C are defined as

$$A = \text{Rot}(\text{IDS}, k_1) \oplus n_1\,, \tag{3}$$

$$B = (\text{Rot}(\text{IDS} \wedge n_1, K_2) \wedge K_1) \oplus n_2\,, \tag{4}$$

$$C = \text{Rot}(R_h(K_1^*), R_h(K_2^*)) \wedge \text{Rot}(R_h(n_1), R_h(n_2))\,, \tag{5}$$

where K_1^* and K_2^* are

$$K_1^* = \text{Rot}(R_h(K_2), R_h(n_1)) \wedge K_1\,, \tag{6}$$

$$K_2^* = \text{Rot}(R_h(K_1), R_h(n_2)) \wedge K_2\,. \tag{7}$$

Upon receiving of $A\|B\|C$ message, the tag extracts random numbers (n_1, n_2) using

$$n_1 = A \oplus \text{Rot}(\text{IDS}, k_1)\,, \tag{8}$$

$$n_2 = B \oplus (\text{Rot}(\text{IDS} \wedge n_1, K_2) \wedge K_1)\,. \tag{9}$$

The tag computes the Seed for recursive hash using Eq. (2). Then, the tag computes its local values of C and C^*, and compares both the values. If both values coincide, then the tag authenticates the reader. Further, the tag computes and sends message

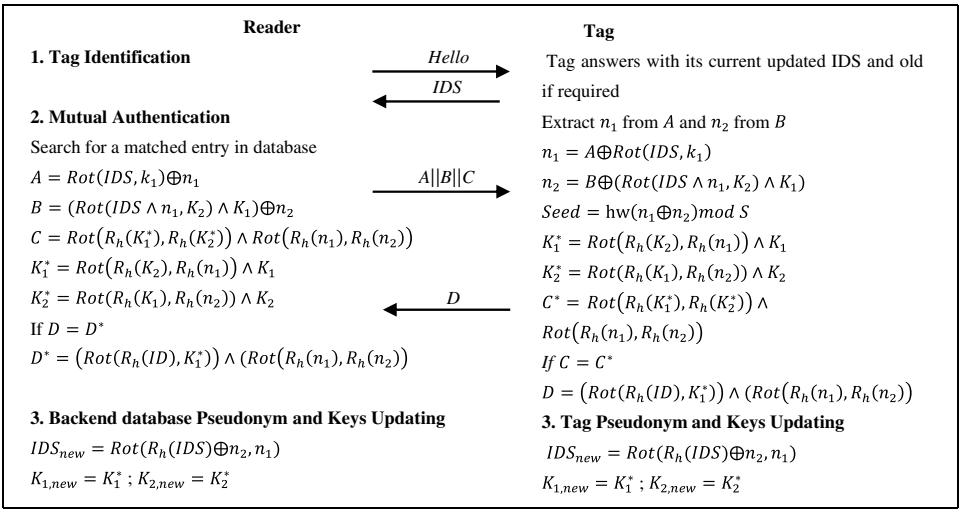

Fig. 2. The RCIA protocol.

D to the reader, where D message is defined as

$$D = (\text{Rot}(R_h(\text{ID}), K_1^*)) \wedge (\text{Rot}(R_h(n_1), R_h(n_2)))\,. \tag{10}$$

The tag will then proceed to pseudonyms updating stage and updates its IDS and keys (K_1, K_2) using the following equations:

$$\text{IDS}_{\text{new}} = \text{Rot}(R_h(\text{IDS}) \oplus n_2, n_1)\,, \tag{11}$$

$$K_{1,\text{new}} = K_1^*\,, \tag{12}$$

$$K_{2,\text{new}} = K_2^*\,. \tag{13}$$

On receiving of D message, the reader computes a local version of D and D^*, and compares both values. If both values match, then the reader also authenticates the tag and updates its pseudonyms and keys using Eqs. (11)–(13).

The quality of randomness of messages A, B, C and D was analyzed using three well-known statistical randomness test suites: ENT,[30] DIEHARD[29] and NIST.[31] Figure 2 shows the detailed working of RCIA protocol.

3. Design and Hardware Architecture

In this section, we present the proposed hardware architecture of the RCIA protocol for the low cost EPC-C1G2 tags. Most of the ultralightweight authentication protocols follow a similar operational (working) model with diverse primitives. The architecture proposed for RCIA can also be used to implement other protocols using 'Rot' primitive (such as SASI,[3] RAPP,[2] the one by Yeh *et al.*,[36] R^2AP,[27] etc.). For efficient implementation (within 4K GE), we have reused the logical components (gates and registers) as much as possible. Figure 3 shows the generic architecture. We

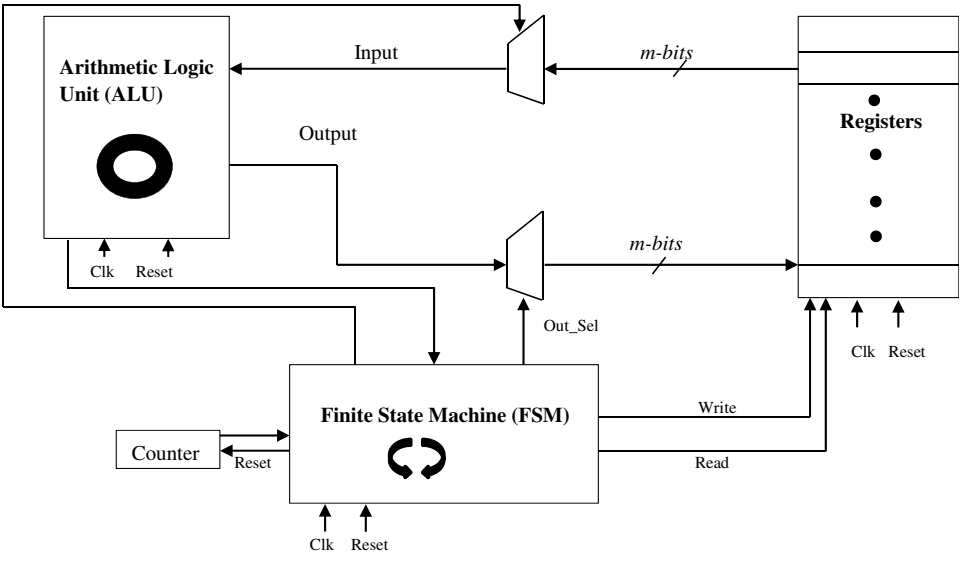

Fig. 3. Hardware architecture of the RCIA protocol.

have also proposed optimal design for non-triangular function (Rot function) as explained in subsequent subsections for optimization. The proposed architecture mainly includes four blocks: registers, arithmetic logic unit (ALU), counter and finite state machine (FSM). The low-level optimized designs and working of the architecture is as follows:

3.1. *Register block*

This block contains all the registers (memory blocks) required to store intermediate computations (results), permanent variables and long-term values. In RCIA, the tag requires 6L dynamic memory to store two entries (old and current) of its pseudonyms ($\mathrm{IDS}_{\mathrm{old}}$ and $\mathrm{IDS}_{\mathrm{new}}$) and keys ($K_{1,\mathrm{old}}, K_{2,\mathrm{old}}, K_{1,\mathrm{new}}$ and $K_{2,\mathrm{new}}$), while 1L static memory to store its permanent ID Moreover, the tag also requires 2L additional dynamic memory to store nonces (n_1, n_2) received from the reader. For hardware optimization and efficiency, we have reused the dynamic registers for intermediate computations (protocol internal operations). Moreover for internal logical operations, we use 10 general purpose (GP) registers $(\mathrm{GP}_0, \mathrm{GP}_1, \ldots, \mathrm{GP}_9)$ of L bits, which hold the intermediate results of ongoing computations and then will be reused after the completion of previous task (computation).

3.2. *ALU block*

The ALU block mainly comprises of bitwise logical operators (protocol specific) and performs the specified computational operations. Hence, the designing of ALU block

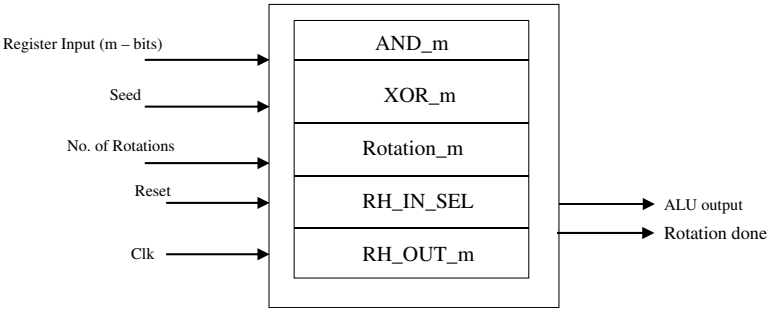

Fig. 4. ALU hardware schematic for the RCIA protocol.

entirely depends upon the protocol (operational) specifications. As far as optimization is concerned, most efforts are concentrated in designing of cost effective ALU block.

In RCIA, ALU mainly performs four logical bitwise operations: AND, XOR, rotation (Rot) and recursive hash (R_h). Figure 4 shows the hardware design of ALU for RCIA protocol.

For optimization, most efforts are concentrated on Rot and recursive hash (R_h) blocks. The remaining operators perform the basic logical operations and there is not much room for optimization. Moreover, the modules have been optimally reused through FSM to reduce the number of gates utilization. The recursive hash (R_h) function is basically the combination of Rot and XOR operations, which requires Seed, memory chunk selection (S_i), XORing and then rotation of 'b' bits of selected memory chunk (S_i) with itself (for efficient hardware implementation $k = 8$ bits.

3.2.1. *Rotation module*

Rotation (Rot) is an essential and the only non-triangular function in ALU, which needs more optimization as remaining functions are basic logical operations (which can be only optimally reused). The $\text{Rot}(X, Y)$ is cyclic left rotation of X according to hamming weight of Y. For $Y = [y_1 y_2 \dots y_m]$, each bit y_i is observed and if $y_i = 1$ then cyclic left rotation on X is performed $(x_{i+1} \dots x_m, x_i)$, otherwise no operation will be performed. The RCIA protocol uses the rotation function with two variations; 8 bits for computation of Recursive hash and m-bit rotations (for other rotation calculations).

Generally, rotation module requires one hamming weight module (computes the hamming weight of the rotor register) and one barrel shifter. For optimization, we implement the rotation module using a simple shifter (left) along with MUX instead of using barrel shifter and hamming weight module. In our implementation, we use two registers: Rotor and To_Rotate. At each clock cycle, the Rotor is shifted left while reading its MSB. If the MSB is '1', the To_Rotate is rotated left; otherwise no

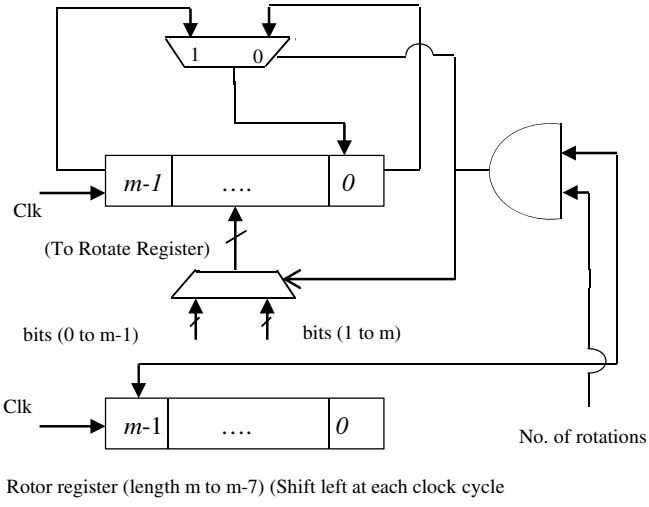

Fig. 5. Rotation module (m-bit).

operation is performed. The process is completed in m cycles avoiding hamming weight computations.

To reduce the size of silicon, both the rotation functions have been implemented within single rotation module. A wire "No. of rotations" selects the partial (8 bits) or whole (m bits) registers for cyclic rotations. If we set No. of rotations = 0, then it rotates 8 bits (recursive hash computations), otherwise it rotates m bits. Figures 5 and 6 represent 8-bit and m-bit rotation modules respectively. This module implements all Eqs. (3)–(11) which incorporate rotation functions.

3.2.2. *Recursive hash module*

Recursive hash (R_h) mainly involves three steps: Decimate the string into 'S' memory chunks, use computed Seed for chunk selection and compute the final recursive hash using the logical operations (XOR and Rot) of selected chunk with the whole string. The detailed computation of recursive hash is presented in Sec. 2. Figure 6 shows the low-level design (architecture) of recursive hash module (RH-IN-SEL and RH-OUT-m). In step-1, MUX will select the 8-bit memory chunk from the n-bit string (according to the Seed). In step-2, selected chunk will be forwarded to the 8-bit rotation and XOR modules. The 8-bit rotation module rotates the 8-bit selected chunk with itself. The XOR module takes XOR between $(S-1)$ copies of 8-bit selected chunk and n-bit string.

Finally, in step-3, the results of the both modules will be applied to de-MUX (control signal, Seed), which will inject (place) the result of rotation module at the specified location (8-bit zeroes) in XORed n-bit string. This module implements Eqs. (5)–(7), (10) and (11) which incorporate recursive hash values of variables.

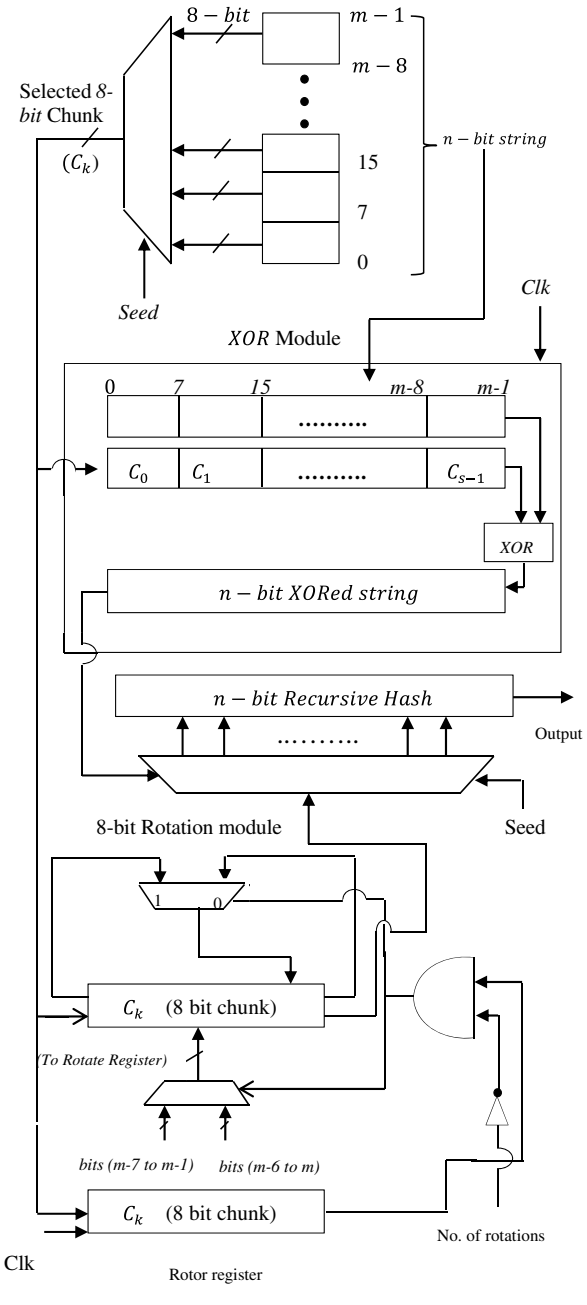

Fig. 6. Recursive hash module.

3.3. *Finite state machine*

Our designed FSM is a simple and traditional state machine which controls the data flow (communication) between various hardware components (ALU and registers). Initially, each tag is in 'Idle' state and after receiving 'Hello' message, the tag moves to the next state 'Send IDS'. The tag then proceeds to the next states (receive_A, receive_B and receive_C) for reception of $A\|B\|C$ messages. The computation of pseudorandom numbers (n_1, n_2) requires six transition states and computation of recursive hash requires two states (XOR and rotation). After comparison of C (received and locally computed values) the tag proceeds to final states "computation of D state" and "pseudonym and keys update states". The FSM optimally reuses the registers and logic gates to reduce the overall chip area, GEs and registers.

4. Circuit Synthesis and Experimentation Results on FPGA and ASIC

In this section, circuit synthesis and experimental results of the proposed design on FPGA and ASIC is presented. We have first described the RCIA protocol in Visual C platform for initial resource estimation and rudimentary working. Then all hardware components of the proposed design were described in VHDL for EPC-C1G2 specified three different bit lengths (32, 64 and 96).

The experimental setting, circuit synthesis and simulation results for both FPGA and ASIC are as follows:

4.1. *Hardware implementation on FPGA*

FPGA based instantiation and synthesis were conducted in XILINX ISE Design Suite 12.3 environment for Spartan-6 and Virtex-6 FPGAs.

We have selected these FPGAs because of their extremely low-power process technologies and optimized resource approximation. Spartan-6 FPGA is built on 45 nm low-power copper process technology with dual FF and efficient six-input LUTs.[35] Virtex-5 FPGA is considerably larger device which is built on 65 nm copper CMOS process technology.[32] Table 2 shows the synthesis report (resource utilization) of the proposed design on Spartan-6 and Virtex-5 FPGAs.

Table 2. Resources utilization of proposed design for various FPGAs.

FPGA	Parameters/Resources	32-bit	64-bit	96-bit
Spartan-6	Number of slice registers	438	889	1221
	Number of slice LUTs	684	1556	1887
	Number of fully used LUT–FFs	205	665	817
Virtex-5	Number of slice registers	552	904	1345
	Number of slice LUTs	602	1326	1793
	Number of fully used LUT–FFs	209	686	837

We have implemented our proposed design for three different bit lengths (32 bits, 64 bits and 96 bits) architectures (for EPC-C1G2 tags). The implementation on Spartan-6 XC6SLX4 device for 32-bit architecture occupied 438 register slices, 684 LUTs and 205 fully used LUT–FF pairs. For 64-bit architecture, registers slices, slice LUTs and fully used LUT–FF pairs are increased to 889, 1556 and 665, respectively. Now, if we further increase the bit length to 96 bits, then inevitably it requires more resources.

For Spartan-6, it requires 1221 register slices, 1887 slice LUTs, 817 fully used LUT–FF pairs and for Virtex-5, 1345 register slices, 1793 slice LUTs and 831 fully used LUT–FF pairs are required. We can see from our synthesis results that increase in bit lengths will increase the resource requirement on FPGAs (which means resources occupancy is independent of FPGA device). So, there is a tradeoff between the level of security robustness and hardware resources requirements, such that if we increase the bit length then it provides more security but requires more resources and vice versa.

Huang *et al.*[24] also implemented and synthesized a similar EPC-C1G2 protocol (CRC-16 based) using PadGen function (for both MOD and XOR) on Virtex-5 XC5VLX30 and Altera Cyclone II. On Virtex-5, XOR scheme requires 599 register slices, 427 slice LUTs and for MOD scheme 643 register slides and 599 slice LUTs are required. However, our proposed architecture (for 32-bit RCIA) requires less number of slice registers (552) and approximately the same amount of slice LUTs (602) on Virtex-5 XC5VLX30 FPGA, which shows RCIA protocol's preeminence in terms of hardware than the PadGen protocol.

4.2. *Hardware implementation on ASIC*

We have used Leonardo Spectrum for ASIC implementation and resource estimation of our proposed architecture (RCIA). TSMC 35 μm library is used for circuit instantiation and synthesis of the proposed design.

For experimental setup, operating frequency was set to 100 KHz for signal clock (as per EPC-C1G2 tag's requirement), while power supply was adjusted to 1.3 V. Number of gates GEs has been used for performance analysis of the proposed architecture. An authentication protocol can only be considered as ultralightweight, if it consumes a maximum of 4000 number of gates for security related tasks.[3] Table 3

Table 3. Hardware results for RCIA protocol (ASIC).

Word length	32-bit	64-bit	96-bit
Number of gates required			
Basic logical operations	3026	5709	7818
Rotation	819	1472	1957
Recursive hash	111	205	314
Total	3956	7386	10,089

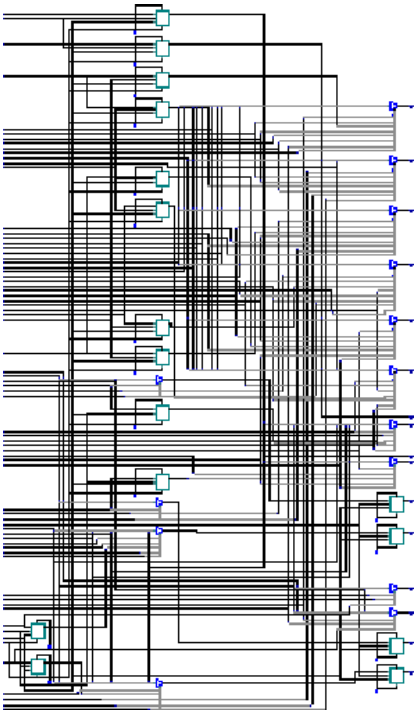

Fig. 7. RTL of proposed (32-bit) architecture using TSMC 0.35 μm.

summarizes the synthesis report (number of gates) of the proposed design for 32-bit, 64-bit and 96-bit lengths. We can see that the area remains below 4000 GEs for 32-bit (which is acquiescent with EPC-C1G2 tags) architecture of the proposed design. For larger bit length, although the security of the protocol enhances, required resources will exceed peripheral of ultralightweight class. Figure 7 shows RTL of proposed architecture for 32-bit word length.

One of the recently published works[25] has implemented two EPC-C1G2 protocols (Burmester–Munilla and Chien–Huang) using ASIC design flow for three different word lengths (32, 64 and 128 bits). Both the protocols incorporate PRNGs in their designs which enhance the diffusion properties of the exchanged messages. Therefore, the authors had put most of the efforts in optimal designing of PRNGs (AKARI-I and AKARI-II). They proposed multiple designs of AKARI-I and AKARI-II PRNGs and compared area requirements, power consumption and throughput for all the designs. After describing hardware components in VHDL, they have used UMC Faraday 90 nm library for synthesis. Like RCIA, the area remains well below (or close) to 4 K for both Burmester–Munilla and Chien–Huang protocols using only 32-bit message length. Hence, these UMAPs also successfully conform to EPC-C1G2 tags for 32-bit word length. For larger bit length, the area grows significantly and hence requires

Table 4. Comparison of RCIA with PRNG based UMAPs.

	Burmester–Munilla				Chien–Huang					
Protocols	AKARI-IA		AKARI-IIA		AKARI-IA		AKARI-IIA		RCIA	
Message length	32-bit	64-bit	32-bit	64-bit	32-bit	64-bit	32-bit	64-bit	32-bit	64-bit
Number of gates	2666	5184	4026	8010	2840	5453	4185	8273	3956	7386

more resources and power. Table 4 compares RCIA protocol with Burmester–Munilla and Chien–Huang protocols in terms of resources requirements (GEs).

5. Reconfigurable Architecture for UMAPs

We can also efficiently implement UMAPs by using reconfigurable architectures. This alternative approach of hardware implementation efficiently tackles the problem of rapid diversifications of the UMAPs and their primitives. Figure 8 presents the generic reconfigurable architecture for the UMAPs based on ARX operations. The generic reconfigurable architecture comprises of four main blocks: Main memory (permanent), secondary memory, FSM and ALU.

- Main memory block: This block stores all the dynamic and static variables such as IDS, keys and intermediate results (including pseudorandom numbers) of the protocol.
- Secondary memory block: After accessing the required data from main memory, the tag forwards the data (variable) to the secondary memory block, which decimates it in (s-bit) memory chunks. The secondary memory block can perform bitwise shifting, cyclic rotations (hamming weight based) and other memory operations (permutation, Sep() and Mer(), etc.). It then forwards the data towards ALU block for basic logical operational computations.

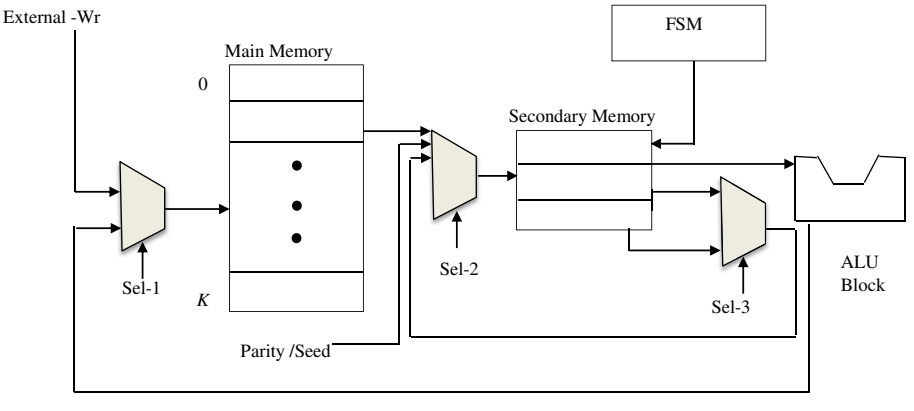

Fig. 8. Generic reconfigurable architecture for UMAPs.

- ALU block: This block comprises of all the protocol specific logical operators (usually T-functions). Moreover, the ALU block and secondary memory block collectively update the pseudonyms and keys after each successful authentication session.
- FSM block: The FSM block mainly controls the data flow (communication) between ALU and secondary memory. It is also responsible for optimal data organization and memory operations within the secondary memory block.

The proposed reconfigurable architecture not only can implement our proposed UMAP but also can implement all other ARX based UMAPs. The reconfigurable architecture presented here is quite similar to the generic architecture presented in Fig. 3 (except the concept of secondary memory), so the hardware utilization on FPGA and ASIC approximately remain the same. (More specifically, for 32-bit architecture, it occupies only 3893 logic gates on ASIC.)

6. Conclusion

One of the major challenges in RFID is to design efficient and cost effective UMAP. EPC-C1G2 tags are passive in nature, which do not possess on-chip battery and hence can support fewer resources for basic routine working. Only a few thousand gates can be allocated to security related tasks, which makes the design of such cryptographic protocols more challenging. The proper hardware implementation of such ultralightweight protocols has been neglected since long; it was unclear that whether such protocols are practically compatible with low cost EPC-C1G2 RFID tags. In this paper, we have addressed this question by exploring the FPGA and ASIC implementations of the newly proposed ultralightweight RFID authentication protocol using recursive hash: RCIA. We have proposed optimized reusable Rot module to perform both m-bit and 8-bit rotations and efficient FSMs for circuit reuse to reduce the silicon area. Our experimental results show that there is a clear tradeoff between circuit area and message length (directly relates to robust security), so optimization of one of them may interrupt the other. We have also proposed a reconfigurable architecture to address the rapid diversification of the standards.

References

1. U. Mujahid, M. Najam-ul-Islam and M. Ali Shami, RCIA: A new ultralightweight RFID authentication protocol using recursive hash, *Int. J. Distrib. Sens. Netw.* **2015** (2015) 642180, doi: 10.1155/2015/642180.
2. Y. Tian, G. Chen and J. Li, A new ultralightweight RFID authentication protocol with permutation, *IEEE Commun. Lett.* **16** (2012) 702–705.
3. H.-Y. Chien, SASI: A new ultralightweight RFID authentication protocol providing strong authentication and strong integrity, *IEEE Trans. Dependable Secur. Comput.* **4** (2007) 337–340.

4. P. Peris-Lopez *et al.*, LMAP: A real lightweight mutual authentication protocol for low-cost RFID tags, *Proc. 2nd Workshop RFID Security*, Austria (2006), pp. 100–112.
5. P. Peris-Lopez *et al.*, EMAP: An efficient mutual-authentication protocol for low-cost RFID tags, *The 1st Int. Workshop Information Security (OTM-2006)*, France (2006), pp. 352–361.
6. P. Peris-Lopez *et al.*, M2AP: A minimalist mutual authentication protocol for low cost RFID tags, *Proc. Ubiquitous Intelligence and Computing* (2006), pp. 912–923.
7. B. Alomair *et al.*, Passive attacks on the class of authentication protocols for RFID, *Information Security and Cryptology: ICISC-2007*, Lecture Notes in Computer Science (Springer, 2007), pp. 102–115.
8. M. Barasz *et al.*, Breaking LMAP, *Workshop RFID Security*, Malaga, Spain (2007).
9. M. Barasz *et al.*, Passive attack against M2AP mutual authentication protocol for RFID tags, *Proc. First EURASIP Workshop on RFID Technology* (2007).
10. P. D'Arco and A. De Santis, On ultralightweight RFID authentication protocols, *IEEE Trans. Dependable Secur. Comput.* **8** (2011) 548–563.
11. H.-M. Sun, W.-C. Ting and K.-H. Wang, On the security of Chien's ultralightweight RFID authentication protocol, *IEEE Trans. Dependable Secur. Comput.* **8** (2011) 315–317.
12. G. Avoine, X. Carpent and B. Martin, Strong authentication and strong integrity (SASI) is not that strong, *Workshop RFID Security and Privacy*, Turkey (2010), pp. 50–64.
13. G. Avoine, X. Carpent and B. Martin, Privacy-friendly synchronized ultralightweight authentication protocols in the storm, *J. Netw. Comput. Appl.* **35** (2012) 826–843.
14. P. Peris-Lopez *et al.*, Advances in ultralightweight cryptography for low-cost RFID tags: Gossamer protocol, *The 9th Int. Workshop Information Security Applications* (2009), pp. 56–68.
15. D. Tagra, M. Rahman and S. Sampalli, Technique for preventing DoS attacks on RFID systems, *Int. Conf. Software, Telecommunications and Computer Networks (SoftCOM)*, Dubrovnik (2010).
16. K. H. Yeh and N. W. Lo, Improvement of two lightweight RFID authentication protocols, *Inf. Assur. Secur. Lett.* **1** (2010) 6–11.
17. Z. Bilal, A. Masood and F. Kausar, Security analysis of ultra-lightweight cryptographic protocol for low-cost RFID tags: Gossamer protocol, *The 12th Int. Conf. Network-Based Information Systems*, Indianapolis, USA (2009), pp. 260–267.
18. M. Zubair, U. Mujahid, M. Najam-ul-Islam and J. Ahmed, Cryptanalysis of RFID ultra-lightweight protocols and comparison between its solutions approaches, *BUJICT J.* **5** (2012) 58–63.
19. M. David and N. R. Prasad, Providing strong security and high privacy in low-cost RFID networks, *Int. Conf. Security and Privacy in Mobile Information and Communication Systems*, Italy (2009), pp. 172–179.
20. J. C. Hernandez-Castro *et al.*, Cryptanalysis of the David–Prasad RFID ultralightweight authentication protocol, *Workshop RFID Security and Privacy*, Turkey (2010), pp. 22–34.
21. D. F. Barrero *et al.*, A genetic Tango attack against the David–Prasad RFID ultra-lightweight authentication protocol, *Expert Syst.* **31** (2014) 9–19.
22. Z. Ahmadian, M. Salmasizadeh and M. Reza Aref, Desynchronization attack on RAPP ultralightweight authentication protocol, *Inf. Process. Lett.* **113** (2013) 205–209.
23. S.-H. Wang *et al.*, Security analysis of RAPP: An RFID authentication protocol based on permutation (2012), Cryptology ePrint Archive Report No. 2012/327, https://eprint.iacr.org/2012/327.

24. Y.-J. Huang *et al.*, Efficient implementation of RFID mutual authentication protocol, *IEEE Trans. Ind. Electron.* **59** (2012) 12.
25. H. Martín *et al.*, Efficient ASIC implementation and analysis of two EPC-C1G2 RFID authentication protocols, *IEEE Sens. J.* **13** (2013) 10.
26. A. Klimov and A. Shamir, New applications of T-functions in block ciphers and hash functions, *Proc. FSE'05*, Vol. 3557 (2005), pp. 18–31.
27. X. Zhuang, Y. Zhu and C.-C. Chang, A new ultralightweight RFID protocol for low-cost tags: R2AP, *Wirel. Pers. Commun.* **79** (2014) 1787–1802.
28. EPCglobal, GS1 EPC global tag data standards version 1.4 (), http://www.epcglobalinc.org/standards/, 2015.
29. G. Marsaglia and W. W. Tsang, Some difficult-to-pass tests of randomness, *J. Stat. Softw.* **7** (2002) 37–51.
30. J. Walker, ENT Randomness Test (1998), http://www.fourmilab.ch/ random/.
31. C. Suresh, J. Charanjit, J. R. Rao and P. Rohatgi, A cautionary note regarding evaluation of AES candidates on smart-cards, *Second Advanced Encryption Standard (AES) Candidate Conf.* (1999), http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm.
32. Xilinx, Virtex-5 Family overview: DS100 (v5.0) (2009), http://www.xilinx.com/support/documentation/data.
33. ATMEL, Understanding the requirements of ISO/IEC 14443 for type B proximity contactless identification cards, Application Note Rev. 2056B-RFID-11/05 (), http://www.atmel.com/images/doc2056, 2015.
34. N. Bagheri *et al.*, Cryptanalysis of RAPP: An RFID authentication protocol: Cryptology ePrint Archive Report No. 2012/701, https://eprint.iacr.org.
35. Xilinx, Spartan-6 Family Overview: DS160 (v2.0) (2011), http://www.xilinx.com/support/documentation/data_Sheets/ds160.
36. K.-H. Yeh, N. W. Lo and E. Winata, An efficient ultralightweight authentication protocol for RFID systems, *Workshop RFID Security and Privacy*, Turkey (2010), pp. 49–60.