




# A low-area design of two-factor authentication using DIES and SBI for IoT security

M. N. Sudha<sup>1</sup> · M. Rajendiran<sup>2</sup> · Mariusz Specht<sup>3</sup>  · Kasarla Satish Reddy<sup>4</sup> · S. Sugumaran<sup>5</sup>

Accepted: 9 August 2021 / Published online: 3 September 2021  
© The Author(s) 2021

## Abstract

Internet of things (IoTs) is an integration of heterogeneous physical devices which are interconnected and communicated over the physical Internet. The design of secure, lightweight and an effective authentication protocol is required, because the information is transmitted among the remote user and numerous sensing devices over the IoT network. Recently, two-factor authentication (TFA) scheme is developed for providing the security among the IoT devices. But, the performances of the IoT network are affected due to the less memory storage and restricted resource of the IoT. In this paper, the integration of data inverting encoding scheme (DIES) and substitution-box-based inverter is proposed for providing the security using the random values of one-time alias identity, challenge, server nonce and device nonce. Here, the linearity of produced random values is decreased for each clock cycle based on the switching characteristics of the selection line in DIES. Moreover, the linear feedback shift register is used in the adaptive physically unclonable function (APUF) for generating the random response value. The APUF–DIES–IoT architecture is analyzed in terms of lookup table, flip flops, slices, frequency and delay. This APUF–DIES–IoT architecture is analyzed for different security and authentication performances. Two existing methods are considered to evaluate the APUF–DIES–IoT architecture such as TFA-PUF-IoT and TFA-APUF-IoT. The APUF–DIES–IoT architecture uses 36 flip flops at Virtex 6; it is less when compared to the TFA-PUF-IoT and TFA-APUF-IoT.

**Keywords** Adaptive physically unclonable function · Data inverting encoding scheme · Internet of things · Linear feedback shift register · One-time alias identity · Two-factor authentication

---

✉ Mariusz Specht  
m.specht@wn.umg.edu.pl

Extended author information available on the last page of the article

## 1 Introduction

Internet of things (IoT) contains huge amount of devices that interconnected over the public Internet [1, 2]. IoT denotes the network of devices, machines, objects and other physical system that has the capacities of computing, embedded sensing and communication. This device supports the systems to sense and transmit the real-time data with the physical world [3, 4]. The high intelligent facility provided by the IoT improves the human's daily lives. Examples of the IoT applications include smart industries, smart home, smart transportation, smart health care and smart cities [5, 6]. The design process and miniaturizing processing techniques are used to improve the IoT. Hence, an improved design process and communication protocol result in high energy storage capacity, high data rate and significant computing capacity [7, 8]. The information exchange between the IoT devices is affected due to the theft, privacy violation and cyber-attacks. In order to overcome the aforementioned issues, the cryptography techniques are accomplished to enable the secure communication [9, 10]. Generally, the architecture of complex software/ hardware is used to create the random number sequences for delivering the public and private keys. Next, the public and private keys are used to accomplish the security in IoT applications [11, 12].

Some of the examples used to provide the security in IoT are proxy-based key agreement protocol [13], malware detection mechanism [14], lightweight block cipher [15], lightweight elliptic curve cryptography [16], PHOTON hash function [17] and so on. Key management is considered as an important constraint in IoT, because of the huge amount of devices and restricted resources in the network. In key management, the frequent generation of modern keys is difficult during key generation phase. Moreover, the frequent generation of modern keys causes higher energy consumption and reduces the device lifetime [18]. Therefore, the development of security is difficult in the resource-constrained hardware platforms such as radio frequency Identification and sensors. The resource constraints of the IoT device are complex cryptographic functions, area and energy that cause a large overhead for IoT devices. For example, the advanced encryption standard (AES) is not appropriate for resource-constrained applications, because of its deficiency in area and energy/power [19]. Subsequently, the physically unclonable function (PUF) circuit is developed as capable hardware security technique for low-overhead security applications, since the working principle of PUF is mainly based on the variation effects of nano-scale device-level process [20]. Moreover, the data is protected using the data flipping in Bose–Chaudhuri–Hocquenghem codes [21]. This flip method is also used to prevent the information loss in the super-resolution technology [22]. The main motivation of this work is to improve the security of the IoT device while minimizing the hardware utilization of the APUF–DIES-IoT architecture.

The major contributions of the paper are given as follows:

- The DIES using SBI is used to generate the random values of the AID, challenge, server nonce and device nonce that are used to improve the security for

the device to server communications. An automatic generation of AID, challenge, server nonce and device nonce is used to decrease the hardware utilization.

- Moreover, the aforementioned 4 values are generated with high randomness for each clock cycle using an appropriate switching between the selection lines.
- Next, the APUF is used to generate response value with higher randomness using LFSR. Accordingly, the TFA and security characteristics are analyzed to evaluate the performances of the APUF–DIES-IoT architecture.
- The APUF–DIES-IoT architecture considers different IoT devices in each clock cycle to accomplish the security over the system. Hence, there is no possibility of acquiring the same response value from the APUF, because each IoT has the different ID and challenge values which are used to obtain the different responses from the APUF

The overall organization of the paper is given as follows: Sect. 2 provides the literature survey about the recent techniques about the security mechanisms in the IoT. The problems found from the existing research and solutions for the problems are stated in Sect. 3. Section 4 provides the detailed description about the APUF–DIES-IoT architecture. The results and discussion of the APUF–DIES-IoT architecture are described in Sect. 5. Finally, the conclusion is made in Sect. 6.

## 2 Related works

The literature survey about the existing techniques related to the PUF and security mechanisms used in the IoT is given in Table 1.

## 3 Problem statement

The problems found from the literature survey and solutions for the problems are described in this section.

In lightweight mutual authentication protocol [24], the authentication latency is less, only when the PUF is processed with lesser message size. For an effective IoT system, the delay between the data transmission is should be less for achieving the faster data transmission. An addition of system components leads to make the IoT system susceptible with security threats [22]. Additionally, the code generated by the TFA-PUF is identical to the all clock cycles. Hence, these code values are easily predicted by the hackers [28]. The manual incorporation of challenge values leads to increase in the hardware utilization that affects the delay and operating frequency. Moreover, the generation of AID, challenge, device nonce and server nonce is same for all iteration. So, it is easy to predict by the hackers during the IoT communication.

**Table 1** Related work

| Author           | Methodology   | Advantage  | Limitation  |
|------------------|---|--|---|
| Zoni et al. [23] | The quasi-cyclic low-density parity-check (QC-LDPC) bit-flipping decoders were presented to obtain the post-quantum cryptography. Next, the architecture of the QC-LDPC was optimized for effectively computing the time-consuming vector matrix multiplications of the bit-flipping decoding process. Subsequently, this decoder was used to choose the resource-performance trade-off without considering the factors of the underlying code. Here, the inputs, the intermediate values and the outputs permitted for managing the underlying codes were saved by using the Block RAMs (BRAM) instead of flip flops | The optimized QC-LDPC was helped to improve the design efficiency  | However, the utilization of BRAM was high in the hardware utilization that increased the area of the overall QC-LDPC architecture |
| Aman et al. [24] | The lightweight mutual authentication protocol was developed for IoT systems using PUF. This mutual authentication protocol was developed for 2 scenarios of IoT systems such as (1) for the communication between server and IoT device and (2) for the communication between two different IoT devices. The PUF-based challenge-response mechanism was used to define the mutual authentication protocol. On the other hand, the mutual authentication protocol has one unique feature that it doesn't require to save any secrets in the IoT devices   | This lightweight mutual authentication protocol was used to minimize storage, communication overhead and computation | However, the authentication latency was minimized only by decreasing the amount of messages transmitted among the devices         |

**Table 1** (continued)

| Author           | Methodology  | Advantage  | Limitation  |
|------------------|--|--|---|
| Amin et al. [25] | <p>The two-factor (smart card and password) user authentication with the RSA cryptosystem was presented in multi-server environments for providing the less complexity and security against the attacks. Meanwhile, the RSA-based multi-server authentication protocol was used to support the mutual authentication and session key agreement among the server and application server. Next, the Burrows–Abadi–Needham (BAN) logic was used to establish the freshness of the session key and accuracy of the mutual authentication</p>   | <p>The RSA-based authentication protocol was effective in terms of complexity when compared to the conventional algorithms. Further, this multi-server authentication protocol was flexible and it contained the user-friendly password change phase</p> | <p>The hardware utilization was not analyzed in this RSA cryptosystem</p>   |
| Qu and Tan [26]  | <p>The two-factor user authentication was developed with key agreement system using elliptic curve cryptosystem (ECC). This ECC-based two-factor user authentication has five different phases such as system initializing phase, the registration phase, login phase, authentication phase and password change phase. Initially, the public and private were computed in the system initializing phase. In the registration phase, appropriate information was submitted to the server, when the user required the authorization. Subsequently, the user inserted the smart card for server login in login phase and mutual authentication was obtained in the authentication phase</p> | <p>Further, the user has updated the password, when the user was required to change the password</p>   | <p>But, the computation cost of the ECC-based two-factor user authentication and key agreement was high when compared to other algorithms</p> |

Table 1 (continued)

| Author               | Methodology   | Advantage   | Limitation   |
|----------------------|---|---|--|
| Xie et al. [27]      | The extended version of the authenticated key exchange (AKE) protocol was presented for supporting the user secrecy and avoiding the offline dictionary attack, desynchronization attack and lost-smart-card attack. Next, the dynamic ID-based Anonymous two-factor AKE protocol was used to support following properties such as adaptive password change, no long-term public key, untraceability/secrecy, forward secrecy and no centralized password storage   | The extended AKE protocol was lightweight in calculations and it required only less amount of message flow during the communication. Therefore, this extended AKE protocol was appropriate for mobile communications and pervasive computing applications | This work analyzed only with less message size; it failed to analyze with higher message size  |
| Gope and Sikdar [28] | The privacy-preserving two-factor authentication (TFA) protocol was developed for IoT devices. This privacy-preserving TFA protocol was used to support the anonymous communication between the IoT device and server installed in the control and data unit. The PUF was considered as one of the authentication factors for the privacy-preserving TFA protocol and this PUF was characterized by a challenge–response pair. This PUF was used to generate the arbitrary string of bits, i.e., response using the bit string as input challenge | The TFA was secured even when the adversary has the physical access to the IoT device. The security features of PUFs were exploited to provide effective the security features for IoT devices  | The code generated by the PUF was identical for all the clock cycles. Hence, the identical code generated by the PUF was easily detected by the unauthorized users |

### 3.1 Solution

The random generation of AID, challenge, device nonce and server nonce at each clock cycle using the selection line switching property of DIES using SBI effectively improves the security against the unauthorized users. Here, the SBM uses the substitution-box (S-box) operation to generate the 8-bit seed value for the DIES by using the substitution process. An automatic generation of AID, challenge, device nonce and server nonce using DIES helps to minimize the number of logical elements during the implementation. Hence, the less amount of hardware utilization increases the speed of the APUF–DIES–IoT architecture that minimizes the delay and increases the operating frequency.

## 4 APUF–DIES–IoT architecture

In APUF–DIES–IoT architecture, the DIES using SBI is used to generate the random values of AID, challenge, device nonce and server nonce at each clock cycle that increases the security against the hackers. Moreover, the APUF is used to generate the random response values using the LFSR. The overall process of the APUF–DIES–IoT architecture is divided into two phases such as setup phase and authentication phase. The process of the setup phase and authentication phase are described in the following section.

### 4.1 Setup phase

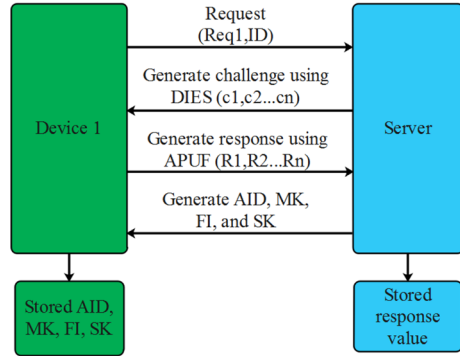
The IoT device produces the identify number (ID) and request during the setup phase as well as the ID and request are connected to the server. Figure 1 shows the operation of the setup phase. More specifically, the green and blue color blocks in Fig. 1 represent the IoT device and server. The values received from the IoT devices are kept in the server and the challenge value is created using the ID. Next, the generated challenge values are associated with the same IoT device. Here, the value of challenge is used to form the response value by using the APUF.

The response values, i.e.,  $R_1, R_2, \dots, R_n$  are kept in the server that are used to generate the synchronization key (SK), master key (MK), fake identity (FI) and one-time alias identity (AID). Subsequently, the generated values of SK, MK, FI and AID are kept in the IoT devices to accomplish the security. The same process of setup process of APUF–DIES–IoT architecture is carried out for all the IoT devices. This setup phase is two-factor authentication, because the SK, MK, FI and AID are generated only when receiving the request from the devices. This process is performed for every IoT devices which improves the security. Additionally, the pairs of fake identity and synchronization keys are generated by the server as shown in Eq. (1).

$$(FD, SK) = \{ (fid_1, k_1), (fid_2, k_2), \dots, (fid_n, k_n) \} \quad (1)$$

where the fake identity is represented as  $fid_1, fid_2, \dots, fid_n$ , synchronization key is represented as  $k_1, k_2, \dots, k_n$  and the  $n$  specifies the number of IoT devices.

Fig. 1 Process of setup phase



### 4.2 Authentication phase

The accessing authorization is precisely provided to the IoT devices, when the device and server nonce is matched during the authentication phase. Figure 2 shows the authentication phase of the APUF–DIES–IoT architecture. Similar to the setup phase, the green and blue color blocks represent the IoT device and server.

In authentication phase, the AID is verified by the random number request and the request message ( $M1$ ) is transmitted to the served for accomplishing the communication, since the request message  $M1$  is stored in the device and it is expressed in Eq. (2).

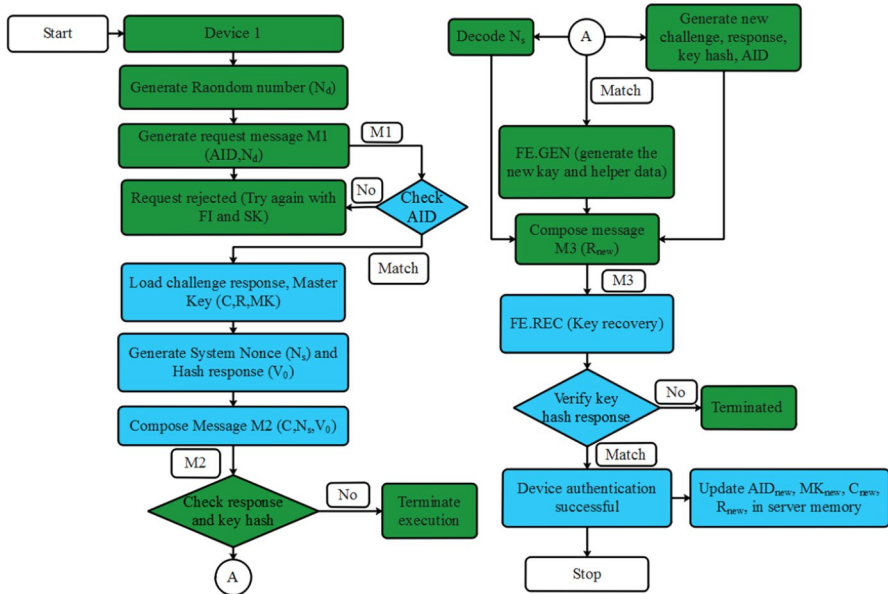


Fig. 2 Flowchart of the authentication phase



$$M1 : \{AID, N_d^*\} \tag{2}$$

where  $N_d^* = N_d \oplus K_{ds}$ . The random number created during the communication is represented  $N_d$  and secret key is represented as  $K_{ds}$ .

The response value, challenge and master key are stored, when the AID is matched in the authentication phase. Otherwise, the generated request is discarded over the IoT device. Subsequently, the hash key response and server nonce ( $N_s$ ) are generated using the server. The server creates the response message ( $M2$ ) as shown in Eq. (3).

$$M2 : \{C, N_s^*, V_0\} \tag{3}$$

where the challenge value is represented as  $C$ ;  $N_s^* = K_{ds} \oplus N_s$  and  $V_0 = h(N_d || K_{ds} || N_s^*)$ ,  $h$  specifies the one-way hash function. Then, the  $M3$  is generated using IoT device when the response is matched during the interaction and this  $M3$  is expressed in Eq. (4).

$$M3 : \{R_{new}^*, V_1, hd^*\} \tag{4}$$

where  $R_{new}^* = k \oplus R'_{new}$ ,  $R'_{new} = \text{RPUF}_{D_i}(C_{new})$ ,  $C_{new} = h(C_i || K_i)$ ,  $hd^* = h(K_{ds} || k || hd)$ ,  $hd = h(K_{ds} || N_s) \oplus hd^*$ ,  $k = \text{FE.Rec}(R, hd)$ ,  $R' = \text{RPUF}_{D_i}(C)$  and  $V_1 = h(N_s || k || R_{new}^* || hd^*)$ . The FE.Gen specifies the helper data generation algorithm,  $R'$  specifies the APUF output,  $hd$  is the helper data,  $k$  is key element and  $V_1$  specifies the key hash response.

The Fuzzy Extractor recovery module is used to address the noise caused in the operation of PUF. The PUF generates the random number at each clock cycle which is compared with key hash response value. Therefore, the IoT device receives the authorization from the server, when the server nonce and device nonce exist in the key hash function. Once the Device 1 is authenticated, the next IoT device (Device 2) performs the same authentication operation. The authentication phase process is performed for each and every IoT devices which APUF values are updated in setup phase itself.

### 4.3 Adaptive PUF

The APUF is designed using LFSR that is identical to the shift register with feedback. The LFSR is mainly used because of its lesser gate computation, lesser computation cost and better statistical properties. The conventional LFSR provides the same random value after certain clock period that affects the security of the IoT system. The reason for using LFSR is to design the APUF to provide the random response value. However, the LFSR provides the same random value after certain clock period. But, the APUF–DIES–IoT architecture considers different IoT devices in each clock cycle to accomplish the security over the system. Hence, there is no possibility of acquiring the same response value from the APUF, because each IoT has the different ID and challenge values which are used to obtain the different responses from the APUF. In the LFSR, the flip flop output is given as feedback to the input of the XOR gate and then the output of the XOR gate is given as input

for the 1<sup>st</sup> flip flop. In the shift register, the initial value is saved that is referred as seed value. This LFSR is used to generate the random sequence of the bits and the feedback output is given to the XOR gate, since the XOR gate is used to improve the confusion property of LFSR. Specifically, the difference between the response values is high by using this XOR gate. Moreover, this LFSR has the capability for generating the possible states at the period of  $N = 2n - 1$ , where  $n$  is the amount of registers. The possible states from the LFSR also exclude all zero state.

In general, the IoT device creates the challenge value and it is given to the server for obtaining the authorization. The conventional PUF module generates the same response for all the clock cycles. Hence, the identical value of responses has the possibility to predict by the unauthorized user. But the APUF used in the proposed method generates the different response values for each and every clock cycle. Equation (5) specifies the challenge input values given to the LFSR. Next, the response output from the LFSR is specified in Eq. (6).

$$\text{challenge}(C) = \{c[7], c[6], c[5], c[4], c[3], c[2], c[1], c[0]\} \quad (5)$$

$$\text{Response}(R) = \{r[7], r[6], r[5], r[4], r[3], r[2], r[1], r[0]\} \quad (6)$$

where the values of  $c[0] - c[7]$  and  $r[0] - r[7]$  represent the challenge values and response values, respectively.

The generation of response for IoT devices is expressed as follows:

```

always @ (posedgeclk)
    if(rst)
        Response(R) = Challenge(C)
    Else
        Response(R) = {r[0]∧r[3], r[7 : 1]}

```

In this APUF, the generation of response is performed at the positive edge of the clock signal. Next, the response is generated from the challenge value, when the clock signal becomes positive edge. Therefore, the variation in the bit pair using APUF at each clock cycle creates the difficulty for response value prediction by unauthorized users. Further, the APUF is used to achieve the secure communication between the devices based on the frequent change of bit position pair.

#### 4.4 Data inverting encoding scheme

In this proposed method, DIES is developed to improve security of lightweight cryptography. The developed DIES uses the confusion property similar to the S-box that processes the 8-bit input data to provide the 8-bit data in output. The size of input and output in DIES are identical, but it provides the different values in output. The input and output of the encoder module are represented as 8 bit, respectively. This DIES increases the randomness in AID, challenge, device nonce and server nonce by independently controlling the odd and even bits of multiplier and multiplicand, the Odd Invert and Even Invert bit, respectively. This will reduce linearity in random data by

comparing the switching activity for the four possible cases of the Full, No, Odd and Even Invert lines (00, 01, 10, 11) and then choosing the value with the smallest switching activity to reduce computational cost. In particular, the input toggling sequences 01 → 10 and 10 → 01 are resulting in 4 times more switching events. The two-phase switching sequence is introduced in order to reduce total power consumption. Encoder module is designed which encodes random number generator based on number of zeroes and ones sequences or its run length. It defines the data to be inverted based on zeroes and ones. It consists of internal modules such as shift register, even counter, odd counter, comparator and inverter shown in Fig. 3 and over all data flipping architecture is shown in Fig. 4.

For the instance, the calculation of AID using DIES is described as follows.

At first, 8-bit input seed value (in) is obtained from the S-box-based inverter and then this 8 bit in value is given as input to the ones calculation for calculating the ones, zeroes, odd and even values of input. In ones calculation process, the counter is zero during reset as well as the counter is incremented by 1 when the counter is less than 9. Consider the values of ones, zeroes, odd and even values are 0000. Subsequently, the ones, zeroes, odd and even values are calculated for each bit of seed value (i.e., totally 8 bits).

The calculation of ones, zeroes, odd and even values for 0th bit, when 0th bit is equal to 1 and 0 is expressed in Eqs. (7) and (8).

$$\begin{aligned}
 \text{ones} &= \text{ones} + 1 \\
 \text{zeroes} &= \text{zeroes} \\
 \text{even} &= \text{even} + 1 \\
 \text{odd} &= \text{odd}
 \end{aligned}
 \tag{7}$$

If the 0th bit is equal to 1, the output is 1010. Otherwise it is equal to 0100.

$$\begin{aligned}
 \text{ones} &= \text{ones} \\
 \text{zeroes} &= \text{zeroes} + 1 \\
 \text{even} &= \text{even} \\
 \text{odd} &= \text{odd}
 \end{aligned}
 \tag{8}$$

Similarly, the calculation of the ones, zeroes, odd and even values for the remaining 7 bits is performed and it is concatenated at the end of ones calculation. Accordingly, the 4-bit values of ones, zeroes, odd and even values are given as input for comparator to obtain the 2 bits of selection line values (Table 2).

Further, the input seed value is modified based on the selection line value that is used to increase the randomness between the AID values generated in each clock cycle. The calculation of AID for the selection line of 00, 01, 10 and 11 is given in Eqs. (9), (10), (11) and (12), respectively.

$$\text{AID} = \{ \sim \text{in} \}
 \tag{9}$$

$$\text{AID} = \{ \text{in}[7], \sim \text{in}[6], \text{in}[5], \sim \text{in}[4], \text{in}[3], \sim \text{in}[2], \text{in}[1], \sim \text{in}[0] \}
 \tag{10}$$

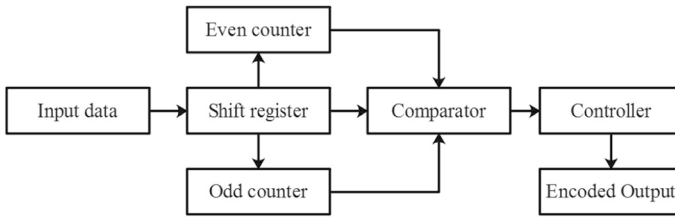


Fig. 3 Encoder module

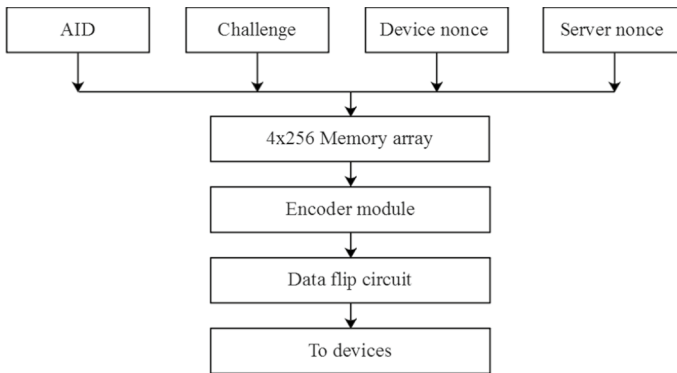


Fig. 4 Architecture of DIES

Table 2 Calculation selection line

| Condition                      | Selection line |
|--------------------------------|----------------|
| Ones > zeroes                  | 00             |
| Ones = zeroes and Even > = odd | 01             |
| Ones = zeroes and Even < odd   | 10             |
| Even > odd                     | 01             |
| Odd > even                     | 10             |

$$AID = \{\sim in[7], in[6], \sim in[5], in[4], \sim in[3], in[2], \sim in[1], in[0]\} \quad (11)$$

$$AID = \sim out \quad (12)$$

Similarly, the challenge, device nonce and server nonce are calculated by using the aforementioned process of DIES. The generated AID, challenge, device nonce and server nonce are used to establish the secure communications between the device and server communication.

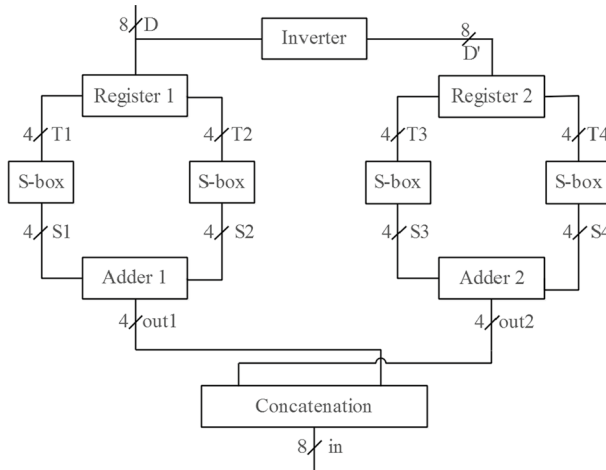


Fig. 5 Architecture of the S-box-based inverter

Table 3 S-box operation

|       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T1    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| S[T1] | C | 5 | 6 | B | 9 | 0 | A | D | 3 | E | F | 8 | 4 | 7 | 1 | 2 |

### 4.5 S-box-based Inverter

In this SBI, an 8-bit seed value is generated for improving the randomness of the AID, challenge, device nonce and server nonce from the DIES. The combination of DIES and SBI is used to generate the unpredictable keys that helps to improve the security among the device to server communication. Figure 5 shows the architecture of the SBI.

Initially, a 8-bit random is considered as an input for this SBI which is represented as  $D$ . Meanwhile, the input 8-bit value is transformed by using the inverter that is represented as  $D'$ . The 8-bit random value  $D$  is stored in the Register 1 and it is truncated into 2 four bits as shown in Eq. (13).

$$\begin{aligned}
 T1 &= D[7 : 4] \\
 T2 &= D[3 : 0]
 \end{aligned}
 \tag{13}$$

Next, these truncated data's  $T1$  and  $T2$  are given to the S-box (substitution-box) that performs the substitution process as shown in Table 3. The S-box generates  $s1$  and  $s2$  for the truncated data of  $T1$  and  $T2$ , respectively.

After completing the S-box process, the data of  $s1$  and  $s2$  are given to the adder for generating the 4-bit value, i.e.,  $out1$ . On the other hand, a 4-bit value of  $out2$  is generated for the  $D'$ . Further, both the  $out1$  and  $out2$  are concatenated together that generates the 8-bit value ( $in$ ) as shown in Eq. (14). Here, the  $out2$  value is taken as MSB and  $out1$  value is taken as LSB for the 8-bit  $in$  value.

$$\text{in} = \{\text{out2}[7 : 4], \text{out1}[3 : 0]\} \quad (14)$$

The designed SBI uses the 8-bit input to provide the 8-bit output value, so the developed SBI is 8-bit design. The generated in value is utilized in the DIES to generate the unpredictable AID, challenge, device nonce and server nonce for each clock cycle and for each plain text. This kind of generation for AID, challenge, device nonce and server nonce creates the difficulty to the hackers which are trying to identify the key values. Therefore, the confidentiality of the data transferred from the device to the server is improved using this APUF–DIES–IoT architecture.

## 5 Simulation setup

The APUF–DIES–IoT architecture is designed and implemented in the Xilinx 14.4 software that is operated with the 4 GB RAM with 500 GB hard disk system. The logical elements used in the authentication and setup phase are designed by using the Verilog language. The hardware utilization of the APUF–DIES–IoT architecture is analyzed by using the Xilinx 14.4 software. Further, the verification of the authentication phase and setup mode is obtained using the Modelsim 10.5 software.

### 5.1 Results and discussion

At first, the setup phase is established to each IoT device and this setup phase is mainly processed using control signals. The clock, enable and reset are enabled as control signals for these devices. The enable and rest signals are varied according to the amount of devices connected to the server. In this phase, totally 100 ns is required to process the single cycle. The 100 ns is separated as 50 ns and 50 ns for the positive and negative clock edge. Moreover, the rising edge and 1 are used to define the edge type and logical value, respectively. The phase control signal of this setup phase is represented as 0. For the remaining control signals, the value is denoted as 1 for operating the setup with acceptable losses.

The setup phase is given to the main block, once the input block is set in the APUF–DIES–IoT architecture. The device generates the ID of the device and request to the server, when the input value is applied into the main block. The challenge value is generated for the devices according to the request. Subsequently, the generated challenge value is processed on the server and this server generated the response for the respective devices. Here, the process of response generation in the server is done by two devices such as PUF and adaptive PUF. The input given to the module is considered as the control signals and challenge values. The conventional PUF generates only the standard response due to its standard challenge value. Hence, there is no variation in the generated response value which is easily hacked by the unauthorized users to process the preserved data. On the other hand, the APUF generates the response values with higher randomness based on its feedback process and random bit pair consideration during APUF XOR operation.

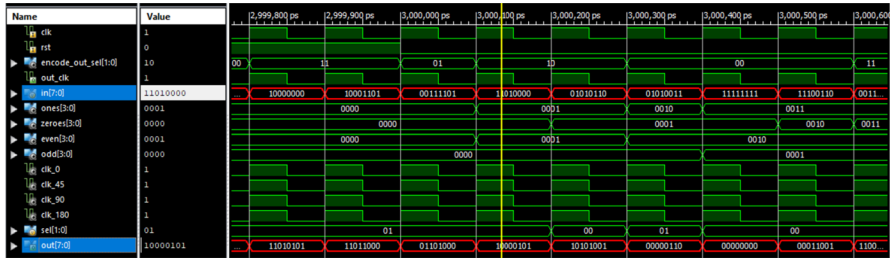


Fig. 6 Simulation waveform of the overall process of DIES

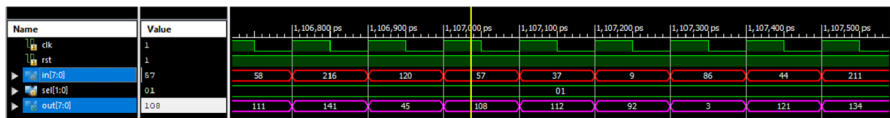


Fig. 7 Simulation waveform of the DIES operation

The IoT devices get the random number that is helpful in authentication phase, once the response is generated in the server. In this APUF–DIES–IoT architecture, the challenge, AID, device nonce and server nonce are generated by using the combination of DIES and SBI. The independent control over the odd and even bits of multiplier and multiplicand, the Odd Invert and Even Invert bit is used to increase the randomness of the challenge, AID, device nonce and server nonce. The switching activity between the selection line is used to minimize the linearity between the values. Next, the simulation waveform of the overall process of DIES using SBI is shown in Fig. 6.

The control signals of Fig. 6 are clk and rst as well as the 8-bit input is termed as in[7 : 0]. From the input given of DIES (11010000), the ones, zeroes, even and odd are calculated to obtain the random AID, challenge, device nonce and server nonce. The values of ones, zeroes, even and odd are 0001, 0000, 0001 and 0000, respectively. Subsequently, the selection line is selected by using the values of ones, zeroes, even and odd. Further, the output obtained from the DIES method is 10000101, i.e., 108. The analysis of the input and output using DIES operation is shown in Fig. 7.

After completing the setup phase, all the IoT device registers their own ID numbers in server and receive an adequate response from the server. Server nonce and device nonce are given as input to the IoT devices. Each IoT device verifies the values of the device and server nonce to verify whether these values exist in the received server and device nonce are not. Subsequently, the new key hash response and helper data are generated by separating the server nonce using IoT device. Accordingly, the generated values are given as input to server to get the authentication. The device nonce and server nonce are presented in the key hash response of the server. Finally, the server provides the authentication for the IoT devices, when the IoT device nonce is exist in the key hash function. The hardware utilization

and security analysis of the APUF–DIES–IoT architecture is given in the following section.

## 5.2 FPGA results and analysis

The hardware utilization of the APUF–DIES–IoT architecture is analyzed with two existing architecture such as TFA-RPUF–IoT architecture and TFA-PUF–IoT architecture [28]. These architectures are designed using Verilog language and the hardware utilization for the aforementioned architectures is given in Table 4. The graphical illustration of the hardware utilization comparison for APUF–DIES–IoT developed in Spartan 6 is shown in Fig. 8. Meanwhile, the comparison of the APUF and conventional PUF [28] is given in Table 5. Next, the graphical illustration of the hardware utilization comparison for PUF and APUF module in Spartan 6 is shown developed in Fig. 9.

From Table 4 and Fig. 7, it is known that the proposed APUF–DIES–IoT architecture achieves better performance when compared to both the TFA-RPUF–IoT and TFA-PUF–IoT [28]. For example, the LUT, slices and flip flops of APUF–DIES–IoT in Spartan 6 FPGA are 10, 10 and 35, respectively, which are less than the TFA-RPUF–IoT and TFA-PUF–IoT [28]. Moreover, the higher frequency of the APUF–DIES–IoT architecture, i.e., 533.67 at MHz shows that it has higher operating speed than the remaining architectures. The hardware utilization of the APUF–DIES–IoT architecture is improved due to its automatic generation of AID, challenge, device nonce and server nonce. Next, Table 5 and Fig. 8 show the analysis of hardware utilization for both the PUF and APUF. The APUF used in the APUF–DIES–IoT architecture utilizes less amount of hardware resources than the conventional PUF architecture [28]. However, the hardware utilization of the Virtex 6 is higher than the Spartan 6, because of requires high amount of logical elements to create the design. Further, Table 6 shows that the DIES uses 5 slices, 7 LUT, 7 flip flops during implementation in Virtex 6 device. Here, the automatic generation of the AID, challenge, server nonce and device nonce is used to reduce the logical elements of the APUF–DIES–IoT architecture than the conventional PUF architecture [28].

Tables 7 and 8 show the analysis of the TFA and security performances for the APUF–DIES–IoT architecture along with two existing architecture such as TFA-RPUF–IoT and TFA-PUF–IoT [28]. Tables 7 and 8 show the comparison of APUF–DIES–IoT architecture with existing researches [25–28] and TFA-RPUF–IoT to analyze the authentication and security features. The clock synchronization, secure algorithm, device security and attacks are evaluated during TFA and the outputs (i.e., Yes or No) are tabulated in the respective portions. Next, the safety against the attacks, two-factor secrecy, mutual authentication and PUF model are analyzed in the security analysis. For both the analysis, the random response for each clock is evaluated for PUF–IoT [28] and APUF–DIES–IoT. The TFA-RPUF–IoT and APUF–DIES–IoT architectures are provided better performances than the existing researches [25–28] because the TFA-RPUF–IoT and APUF–DIES–IoT generate the random input data even when the input remains

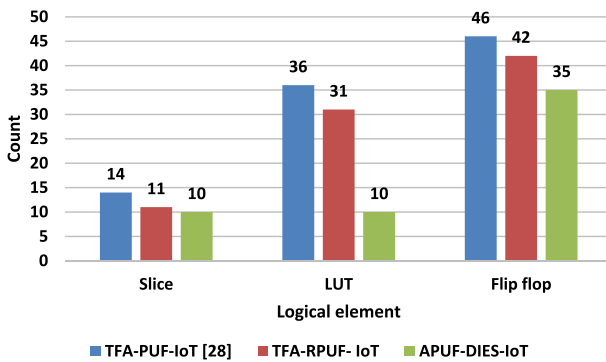


**Table 4** Analysis of hardware utilization for different security architectures

| Target Family | Device    | Speed and package | Architecture     | Slice | LUT | Flip flop | Frequency (MHz) | Delay (ns) |
|---------------|-----------|-------------------|------------------|-------|-----|-----------|-----------------|------------|
| Spartan 6     | XC6SLX9   | - 3 and CSG324    | TFA-PUF-IoT [28] | 14    | 36  | 46        | 441.92          | 2.263      |
|               |           |                   | TFA-RPUF- IoT    | 11    | 31  | 42        | 510.06          | 1.961      |
|               |           |                   | APUF-DIES-IoT    | 10    | 10  | 35        | 533.67          | 1.237      |
| Virtex 6      | XC6VCX75T | -2 and FF484      | TFA-PUF-IoT [28] | 17    | 34  | 46        | 713.59          | 1.401      |
|               |           |                   | TFA-RPUF- IoT    | 10    | 25  | 42        | 739.61          | 1.352      |
|               |           |                   | APUF-DIES-IoT    | 9     | 21  | 36        | 742.45          | 1.021      |

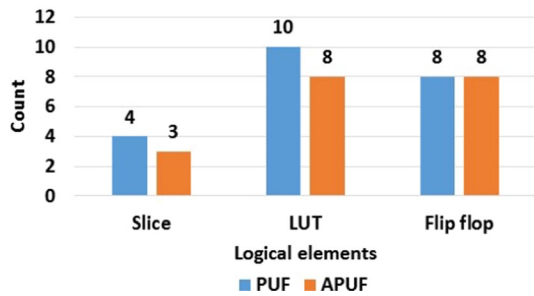
**Table 5** Analysis of hardware utilization for conventional PUF and APUF

| Target family | Device    | Speed and package | Module | Slice | LUT | Flip flop | Frequency (MHz) | Delay (ns) |
|---------------|-----------|-------------------|--------|-------|-----|-----------|-----------------|------------|
| Spartan 6     | XC6SLX9   | – 3 and CSG324    | PUF    | 4     | 10  | 8         | 663.46          | 1.507      |
|               |           |                   | APUF   | 3     | 8   | 8         | 696.45          | 1.436      |
| Virtex 6      | XC6VCX75T | – 2 and FF484     | PUF    | 3     | 10  | 8         | 1225.41         | 0.816      |
|               |           |                   | APUF   | 6     | 9   | 8         | 1243.54         | 0.804      |



**Fig. 9** Hardware utilization comparison of PUF and APUF for Spartan 6

**Fig. 8** Hardware utilization comparison of APUF–DIES–IoT for Spartan 6



**Table 6** Analysis of hardware utilization for DIES used in the APUF–DIES–IoT architecture

| Parameters      | Spartan 6 | Virtex 6 |
|-----------------|-----------|----------|
| Slice           | 5         | 5        |
| LUT             | 10        | 7        |
| Flip flop       | 10        | 7        |
| Frequency (MHz) | 713.73    | 1301.73  |
| Delay (ns)      | 1.2136    | 0.762    |

**Table 7** Analysis of TFA performances for different security architectures of IoT

| Security property                          | Amin [25] | Han [26] | Xie [27] | TFA-PUF-IoT [28] | TFA-RPUF-IoT | APUF-DIES-IoT |
|--|-----------|----------|----------|------------------|--------------|---------------|
| Resilience to the impersonation attack     | Yes       | Yes      | Yes      | Yes              | Yes          | Yes           |
| Anonymity and untraceability               | Yes       | No       | Yes      | Yes              | Yes          | Yes           |
| Resilience to the password guessing attack | No        | Yes      | Yes      | Yes              | Yes          | Yes           |
| Prevents clock synchronization problem     | No        | Yes      | No       | Yes              | Yes          | Yes           |
| Device security                            | No        | No       | No       | Yes              | Yes          | Yes           |
| Deployed security algorithm                | ECC       | ECC      | ECC      | PUF and FE       | RPUF-FE      | DIES          |
| Random response for every clock cycle      | No        | No       | No       | No               | Yes          | Yes           |

**Table 8** Analysis of security performances for different security architectures of IoT

| Comparison matrices                   | Aman [24] | TFA-PUF-IoT [28] | TFA-RPUF-IoT | APUF-DIES-IoT |
|---------------------------------------|-----------|------------------|--------------|---------------|
| Mutual authentication                 | Yes       | Yes              | Yes          | Yes           |
| Two-factor secrecy                    | No        | Yes              | Yes          | Yes           |
| Privacy of the IoT devices            | No        | Yes              | Yes          | Yes           |
| Consideration of noise in the PUF     | No        | Yes              | Yes          | Yes           |
| Protection against physical attacks   | Yes       | Yes              | Yes          | Yes           |
| Random response for every clock cycle | No        | No               | Yes          | Yes           |

same for all clock cycle. The code generated by all clock cycles is same in the TFA-PUF [28]. Therefore, the code generated by the TFA-PUF can easily predict by the hackers [28]. However, the random values such as AID, challenge, device nonce and server nonce generated by the DIES are used to improve the security against the hackers. The APUF-DIES-IoT architecture also obtains lesser hardware utilization than the TFA-RPUF-IoT. Therefore, the APUF-DIES-IoT architecture is referred as better when compared to the existing security mechanisms developed in the IoT.

### 5.3 Security analysis

The different security analysis is evaluated for this APUF-DIES-IoT architecture. The APUF-DIES-IoT architecture has higher confidentiality than the existing TFA-PUF-IoT architecture [28] and TFA-RPUF-IoT architecture.

#### 5.3.1 Session Key agreement

The IoT device and server share the same session key, once the mutual authentication is completed in the IoT. Here, the side channel attack affects the transmission line during the data transmission. If the side channel attack occurred in the APUF-DIES-IoT architecture, the secret key agreement is not encrypted based on the session key corruption. The server doesn't give the authentication for the IoT devices, even when the secret key is changed in the IoT. Hence, the proposed APUF-DIES-IoT architecture has the capacity to provide the session key agreement.

## 6 Conclusion

In this paper, the combination of DIES and SBI is introduced to provide the random values of AID, challenge, server nonce and device nonce for accomplishing the secure communication. The security is additionally improved based on the random

seed value generated by using the SBI. The selection line switching property helps to increase the randomness of AID, challenge, server nonce and device nonce between all clock cycles. Additionally, the LFSR is used in the APUF to generate the random response for every clock cycle. The combination of APUF and DIES effectively improves the security in the IoT system. Hence, the communication between the IoT devices to the server is secured by using the proposed APUF–DIES–IoT architecture. Moreover, the automatic generation of the AID, challenge, server nonce and device nonce is used to minimize the logical elements used in the APUF–DIES–IoT architecture. Accordingly, the delay and operating frequency of the APUF–DIES–IoT architecture are improved during the server to device communication. From the performance analysis, it is known that the proposed APUF–DIES–IoT architecture has better performance than the conventional architectures such as TFA–PUF–IoT and TFA–RPUF–IoT. The proposed APUF–DIES–IoT architecture designed in the Virtex 6 uses 36 flip flops; it is less when compared to the conventional TFA–PUF–IoT and TFA–RPUF–IoT architectures. In the future, different optimized architectures will be implemented to reduce the hardware utilization and improve the security.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.


## References

1. Pan C (2021) Design of sports course management system based on Internet of things and FPGA system. *Microprocess Microsyst* 80:103357
2. Khan AF, Anandharaj G (2020) A multi-layer security approach for DDoS detection in Internet of things. *Int J Intell Unmanned Syst* 9(3):178–191
3. Majumder A, Saha S, Chakrabarti A (2020) EAAM: energy-aware application management strategy for FPGA-based IoT-Cloud environments. *J Supercomput* 76:10258–10287
4. Khan AF, Anandharaj G (2020) Ahkm: an improved class of hash based key management mechanism with combined solution for single hop and multi hop nodes in iot. *Egypt Inf J* 22(2):119–124
5. Taher BH, Jiang S, Yassin AA, Lu H (2019) Low-overhead remote user authentication protocol for IoT based on a fuzzy extractor and feature extraction. *IEEE Access* 7:148950–148966
6. Satpathy S, Mohan P, Das S, Debbarma S (2020) A new healthcare diagnosis system using an IoT-based fuzzy classifier with FPGA. *J Supercomput* 76:5849–5861
7. Farooq U, Hasan NU, Baig I, Shehzad N (2019) Efficient adaptive framework for securing the Internet of things devices. *EURASIP J Wirel Commun Netw* 2019(1):210
8. Khan AF, Anandharaj G (2021) A cognitive energy efficient and trusted routing model for the security of wireless sensor networks: CEMT. *Wirel Pers Commun*, pp. 1–11
9. Kim HK, Sunwoo MH (2019) Low power AES Using 8-bit and 32-bit datapath optimization for small Internet-of-Things (IoT). *J Signal Process Syst* 91(11–12):1283–1289

10. El Hadj Youssef W, Abdelli A, Dridi F, Machhout M (2020) Hardware implementation of secure lightweight cryptographic designs for IoT applications. *Secur Commun Netw* 2020:1–13
11. Stanchieri GDP, De Marcellis A, Palange E, Faccio M (2019) A true random number generator architecture based on a reduced number of FPGA primitives. *AEU Int J Electron Commun* 105:15–23
12. Ferozkhan AB (2021) The embedded framework for securing the Internet of things. *J Eng Res* 9(2):139–148
13. Braeken A, Liyanage M, Jurcut AD (2019) Anonymous lightweight proxy based key agreement for IoT (ALPKA). *Wirel Pers Commun* 106(2):345–364
14. Takase H, Kobayashi R, Kato M, Ohmura R (2020) A prototype implementation and evaluation of the malware detection mechanism for IoT devices using the processor information. *Int J Inf Secur* 19(1):71–81
15. Biswas A, Majumdar A, Nath S, Dutta A, Baishnab KL (2020) LRBC: a lightweight block cipher design for resource constrained IoT devices. *J Ambient Intell Humaniz Comput* 67(3):3563–3579. <https://doi.org/10.1007/s12652-020-01694-9>
16. Lara-Nino CA, Diaz-Perez A, Morales-Sandoval M (2020) Lightweight elliptic curve cryptography accelerator for Internet of things applications. *Ad Hoc Netw* 103:102159
17. Al-Shatari MOA, Hussin FA, Abd Aziz A, Witjaksono G, Tran XT (2020) FPGA-based lightweight hardware architecture of the PHOTON hash function for IoT edge devices. *IEEE Access* 8:207610–207618
18. McGinthy JM, Michaels AJ (2019) Further analysis of PRNG-based key derivation functions. *IEEE Access* 7:95978–95986
19. Singh A, Chawla N, Ko JH, Kar M, Mukhopadhyay S (2018) Energy efficient and side-channel secure cryptographic hardware for IoT-edge nodes. *IEEE Internet Things J* 6(1):421–434
20. Chien WC, Chang YC, Tsou YT, Kuo SY, Chang CR (2020) STT-DPSA: digital PUF-based secure authentication using STT-MRAM for the Internet of things. *Micromachines* 11(5):502
21. Zhang R, Sachnev V, Botnan MB, Kim HJ, Heo J (2012) An efficient embedder for BCH coding for steganography. *IEEE Trans Inf Theory* 58(12):7272–7279
22. Manabe T, Shibata Y, Oguri K (2018) FPGA implementation of a real-time super-resolution system using flips and an RNS-based CNN. *IEICE Trans Fundam Electron Commun Comput Sci* 101(12):2280–2289
23. Zoni D, Galimberti A, Fornaciari W (2020) Efficient and scalable FPGA-oriented design of QC-LDPC bit-flipping decoders for post-quantum cryptography. *IEEE Access* 8:163419–163433
24. Aman MN, Chua KC, Sikdar B (2017) Mutual authentication in IoT systems using physical unclonable functions. *IEEE Internet Things J* 4(5):1327–1340
25. Amin R, Islam SK, Khan MK, Karati A, Giri D, Kumari S (2017) A two-factor RSA-based robust authentication system for multiserver environments. *Secur Commun Netw* 2017:1–15
26. Qu J, Tan XL (2014) Two-factor user authentication with key agreement scheme based on elliptic curve cryptosystem. *J Elect Comput Eng* 2014:1–6
27. Xie Q, Wong DS, Wang G, Tan X, Chen K, Fang L (2017) Provably secure dynamic ID-based anonymous two-factor authenticated key exchange protocol with extended security model. *IEEE Trans Inf Forensics Secur* 12(6):1382–1392
28. Gope P, Sikdar B (2018) Lightweight and privacy-preserving two-factor authentication scheme for IoT devices. *IEEE Internet Things J* 6(1):580–589

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

## Authors and Affiliations

**M. N. Sudha**<sup>1</sup> · **M. Rajendiran**<sup>2</sup> · **Mariusz Specht**<sup>3</sup>  · **Kasarla Satish Reddy**<sup>4</sup> · **S. Sugumaran**<sup>5</sup>

M. N. Sudha  
mnsudhairtt@gmail.com

M. Rajendiran  
mrajendiran@gmail.com

Kasarla Satish Reddy  
ksathishphd2017@gmail.com

S. Sugumaran  
sugumaran.s@vishnu.edu.in

- <sup>1</sup> Department of Information Technology, Institute of Road and Transport Technology, Erode, India
- <sup>2</sup> Department of Information Technology, Mahendra Institute of Technology, Namakkal, Tamil Nadu, India
- <sup>3</sup> Department of Transport and Logistics, Gdynia Maritime University, Gdynia, Poland
- <sup>4</sup> Bangalore Institute of Technology, Bangalore, India
- <sup>5</sup> Department of ECE, Vishnu Institute of Technology, Bhimavaram, AP, India