International Journal of System Dynamics Applications
Volume 11 • Issue 2

Multi-Layer and Clustering-Based Security Implementation for an IoT Environment

Deena Nath Gupta, Jamia Millia Islamia, India
https://orcid.org/0000-0001-6323-411X

Rajendra Kumar, Jamia Millia Islamia, India

ABSTRACT

IoT devices have many constraints related to computation power and memory. Many existing cryptographic algorithms of security could not work with IoT devices because of these constraints. Since the sensors are used largely to collect the relevant data in an IoT environment, and different sensor devices transmit this data as useful information, the first thing that needs to be secured is the identity of devices. The second most important thing is the reliable information transmission between a sensor node and a sink node. While designing the cryptographic method in the IoT environment, programmers need to keep in mind the power limitation of the constraint devices. Mutual authentication between devices and encryption-decryption of messages needs some sort of secure key. In the proposed cryptographic environment, there will be a hierarchical clustering, and devices will get registered by the authentication center at the time they enter the cluster. The devices will get mutually authenticated before initiating any conversation and will have to follow the public key protocol.

KEYWORDS

Cryptographic Protocols, Device-to-Device Communication, Hierarchical Clustering, Information Security, Internet of Things, Lightweight Cryptography, Mutual Authentication, Random Number Generation

INTRODUCTION

Random numbers play an essential role in cryptographic applications. The journey started long back in 1983 when the scientists from the University of California presented their work on random number generation. This work is commonly known as the Blum Sub generator (Blum, 1986). IoT is also known as the constrained environment because devices used in this network are low powered. These devices are not capable of performing complex mathematical calculations because of the large number of Circuit Gates, more than 2000 Gate Equivalent (GE), used. The EPCglobal® restricts the GE to be less than 2000 for the use in constrained devices (GS1, 2013). Hence the researcher needs some sort of less complicated procedure for their calculations. Mathematicians find that the computational power required for shift operations needs much lower power than the other mathematical operations. The researcher can use any of the listed shift operations (left shift, right shift, circular shift, etc.) to permute their bit sequences (GUPTA et al., 2020).

DOI: 10.4018/IJSDA.20220701.oa3

Some test suites are there to examine the randomness of generator functions. One can test RNG work on Diehard battery designed by Marsaglia or on TestU01 suite with 6 test batteries (Small Crush, Crush, Big Crush, Alphabit, and Rabbit batteries and a pseudo-NIST battery) designed by L'ecuyer and Simard, or on the NIST test suite having 15 tests. The National Institute of Statistics and Technology released SP800-90 (a, b, c) recently (Rukhin et al., 2010).

Here, the authors are following the specifications given by the National Institute of Standards and Technology. A uniform and independent distribution of both the digits (zero and one), is the prime requirement from an ideal random number generator. A random number generator used in current cryptographic applications is a sequence of 26-bit or 32-bit discrete values. One can further divide random number generators into two parts; True Random Number Generators (TRNGs) and Pseudo Random Number Generators (PRNGs). The natural source of randomnesses like oscillators or thermal noise is in use for the generation of exact random numbers. These sources are unpredictable because of entropy. This variation produces a random output. Pseudo Random Number Generators (PRNGs) or Deterministic Random Number Generators (DRNGs) are purely based on programming (Gupta & Kumar, 2019).

Peris et al. presented their work in 2007 in which they generated some random numbers and then applied genetic programming on them to create a large number of sequences. They, however, are not that efficient in terms of circuit gate count but somehow manage to be less than 2000 GE (minimum requirement to name any algorithm lightweight) (Peris-Lopez et al., 2009). In 2008, Che et al. proposed a new method of generating random numbers by using valid random physical sources, like low-frequency oscillators and thermal noise generators. As they create output bits using very little power, one can use it as a component in his/her RNG design (Che et al., 2008). Electronic Product Code (EPCglobal®) issues some specifications regarding the manufacturing details of tags and readers. One should follow these restrictions to make their security design compatible with lightweight cryptographic applications. Any random number generator should go through the NIST suite to test their randomness. Many other tests are also available like the ENT test, David Sexton's battery, Diehard suite to check the randomness in obtained sequences.

IoT devices are having many constraints related to computation power and memory etc. Many existing cryptographic algorithms of security could not work with IoT devices because of these constraints. Since the sensors are used in large amounts to collect the relevant data in an IoT environment, and different sensor devices transmit these data as useful information, the first thing that needs to be secure is the identity of devices. The second most important thing is the reliable information transmission between a sensor node and a sink node. While designing the cryptographic method in the IoT environment, programmers need to keep in mind the power limitation of the constraint devices. Mutual authentication between devices and encryption-decryption of messages need some sort of secure key. In the proposed cryptographic environment, there will be a hierarchical clustering, and devices will get registered by the authentication center at the time they enter the cluster. The devices will get mutually authenticated before initiating any conversation and will have to follow the public key protocol.

The organization of the study is as follows—section 2 surveys related works based on different technologies of the lightweight security scheme. Proposed public key protocol is described in section 3. Section 4 presents the layering and clustering of devices in an IoT environment. Section 5 describes the proposed method having four modules, namely, the BiBiSeG module, the RandKeyGen module, the KeyConversion module, and the EncDec module. In section 6, the authors show the implementation of the proposed method. It includes device registration, mutual authentication, public key protocol, and hierarchical clustering. Section 7 presents the experimental setup and results. Section 8 shows the security analysis. Section 9 describes the countermeasure of expected threats. Section 10 gives the conclusion and future work.

RELATED WORKS

While searching for new methods to produce random binary bit sequences, authors explore many technologies that they found useful in the generation process of random binary bit sequences (Bussi et al., 2016; Gao et al., 2014; Leonard & Jackson, 2015; Poorghanad et al., 2008; Salustowicz & Schmidhuber, 1997; Stipcevic & Rogina, 2006; Vasyltsov et al., 2008; Wu & O'Neill, 2010; Naugle et al., 2017; Auxilia et al., 2020). Researchers can generate random bit sequences from any event, like from rolling of dice or from flipping a coin. In computational theory, the researcher creates random sequences either by using linear feedback shift registers, or by using genetic programming, or by transforming a circuit from its meta-stable to bi-stable state, or by performing simple mathematical operations or anything else. The author segregates the work from different researchers of different fields to get a clear picture of every method. The authors also presented performance analysis and an impact in cryptographic applications of different techniques. Many techniques exist to generate random binary bit sequences. Still, the author chooses only the methods passing the lightweight cryptographic criteria from NIST and constraint device applicability criteria from EPC global. Section 'A' presents some works based on the use of linear feedback shift register, section 'B' presents the works on genetic programming. Random number generation also uses the concept of digital circuit artifacts; section 'C' shows the work related to digital circuit artifacts. Section 'D' presents the works based on mathematical operations and generalized feedback shift registers.

PRNGs Based on Polynomial Fitted LFSR

Melia-Segui et al., in 2011, performed an attack on the work of Che et al. scheme by exploring some vulnerability. Only a little knowledge about the LFSR parameter was sufficient for the successful attack on Che et al. (Melià-Seguí et al., 2011). The authors claimed to attack with only 250 bits. In their presented work, authors mask the linearity of the LFSR by using the TRNG bits, unlike the feedback output used in Che et al. They used eight different polynomials for regeneration. The proposed PRNG of Melia-Segui et al. requires a Gate Count of 761. The gate size is nearly half of the work of Peris et al., i.e., 1566. They tested their generated bit sequences on NIST for randomness and found them within the specified range (Melia-Segui et al., 2010).

In 2013, Melia-Segui et al. next proposed some improvements to their previous work (Melià-Seguí et al., 2013). Here they used a set of accurate random sources for selecting the polynomials in a non-linear fashion. At the next level, they perform a logical operation on generated sequences from LFSR before using it as finally made random numbers. They showed a table that contains LFSR sizes varying from 16 bits to 64 bits. The minimum required gate size for their design was 439.1 only. The gate size was approximately 60% of their previous work.

In the same year, Kalikinkal Mandal et al. suggested the use of a mathematical function WG transformation for the generation of random numbers using Non-linear Feedback Shift Registers (Mandal et al., 2016). Although it contains a high degree of accurate calculations, it is suitable for the IoT application, which requires high security. Their design, Warbler, passed all the tests contained in the NIST test suite. The GE of Warbler was approximately 760. The gate requirement was doubled in comparison to Melia-Segui's new work.

In 2015, Jiagang Chen et al. presented their work in IEEE Trustcom (Chen et al., 2015). The authors reviewed the work of Che et al. and suggested an improved lightweight PRNG for low-cost RFID tags. They first proposed an attack on J3Gen and highlighted some weaknesses. Using the same set of polynomials as in J3Gen, they suggested the use of three-bit input from TRNG for polynomial selection. They claimed it to output an entirely random sequence with 50% chances of occurrence for both bits. In the same work, they proposed a second method in which they divide the polynomials into two parts, the first set contains three polynomials, and the second contains five polynomials. The authors designed two separate polynomials for these sets. Programmers perform XOR operation on

the outputs of these two polynomials before generating the final random sequences. After that, the programmer tested for randomness over NIST. They claim it to be more secure than J3Gen, but the authors found nothing about their GE usage.

PRNGs Based on Genetic Programming

Koza, in 1991, worked on producing a sequence of random binary digits (Koza, 1991). It creates the figures by converting a series of consecutive integers by using genetic programming. Concerning that particular measure, it exceeded the performance of the other randomizers. For fitness function, Koza uses the Shannon entropy for information, and they developed the code that accepts the series of integers and outputs a random binary digit sequence.

Marco Tomassini et al., in 1999, applied cellular automata (CA) for generating the pseudo random sequences (Tomassini et al., 1999). They focused on cellular automata, which can produce high-quality random numbers rapidly, and it is better in terms of hardware implementation. They verified CA using an extended battery of tests. Marco Tomassini et al. claimed that non-uniform CAs are better than uniform CAs without time spacing. It is the fastest method of producing random numbers. The strength is the RNG of choice, though they are somewhat inferior to linear congruential and lagged-Fibonacci ones, the quality of the random number sequences produced is quite high and is sufficient for many applications.

Philip Leonard et al. used Shannon's theory of information as to their fitness function and started using single node genetic programming for generating high entropy RNGs in 2015 (Leonard & Jackson, 2015). They claimed it to be six times faster and two times more efficient than Koza's model. They found that single node genetic programming produces better results than standard genetic programming. The code written for this method is more than five times smaller than the systems written with standard genetic programming. The NIST suite of randomness shows that the generated sequences are having properties similar to PRNGs.

In 2016 Stjepan Picek et al. coined the term "Cartesian genetic programming" for the first time. In their design, they used a Deterministic Random Number Generator (Picek et al., 2019). They identified some applications where true randomness was not needed. Like in masking, they are just covering the data to protect them from side-channel attacks. Authors used some fitness functions for their genetic programming to generate the bit sequences that behave like PRNGs. They claimed that their fitness function produces fast random digits, and also this method is not dependable on costly hardware.

Chlumecky et al., in 2017, described the methodology and software for the optimization of rainfall-runoff modeling using a genetic algorithm (GA) with a newly prepared concept of a random number generator (HRNG), which is the core of the optimization (Chlumecký et al., 2017). The GA estimates model parameters using evolutionary principles, which require a quality number generator. The new HRNG generates random numbers based on hydrological information, and it provides better numbers compared to pure software generators. They also focused on improving the internal structure of the GA. HRNG provides a stable trend in the output quality of the model, despite various configurations of the GA. Hence the HRNG speeds up the calibration of the model and offers an improvement of rainfall-runoff modeling.

Cem et al., in 2018, produced PRNG with genetic programming methods using entropy calculation as the fitness function (Kösemen et al., 2018). It satisfies the requirements of the NIST and EPCGen2 standards. Various works generated PRNGs with different GP and fitness methods, but very few of them can practically pass the lightweight criteria, like WISP passive RFID tags. The PRNG made by them is tested on real hardware and analyzed in terms of resource and time consumption.

PRNGs Based on Digital Circuit Artifacts

In 2003, Michael Epstein et al. suggested a new way of generating random numbers (Epstein et al., 2003). They used digital circuits. Some circuits are unstable oscillators, while some exhibit meta-

stability. Using nine distinct designs, they prepare a prototype and test it over a breadboard. They succeed in finding the randomness in obtained sequences. They were not that much as successful as Diehard Suite is concerned, yet they output considerable random sequences. This generator, as claimed by the author, is stable in the term of operating voltage.

In 2008, Ihor Vasyltsov et al. presented a fast digital TRNG based on a meta-stable ring oscillators (Vasyltsov et al., 2008). Jitter based generation was conventional, but this one works for ring oscillator. Its design needed only a digital component. In reasonable condition, they achieved the throughput as high as 140 Mbits/sec, but with some compensation, they managed it to nearly 50 Mbits/sec. The authors claimed to pass FIPS 140-1/2 test, AIS.31 Class P1 / P2 test, and NIST STS test.

In 2010, Wu et al. presented four low-cost circuits using different hardware components (Wu & O'Neill, 2010). They implement XOR gate, lookup table, and multiplexer & inverter on FPGA (field-programmable gate array), and they passed Diehard and NIST test suites. Fourth circuit is using four transistors to generate 80 MB sequences for the Diehard test and 1 GB sequences for the NIST test suite. It passes all the tests but one. The authors claim that their cost is less than the other existing PRNG of this type.

PRNGs Based on the Generalized Feedback Shift Register

In 1992, Matsumoto and Kurita presented their work on a twisted GFSR generator. They found drawbacks in Lewis and Payne PRNG (Matsumoto & Kurita, 1992). Lewis and Payne have a time-complex initialization scheme, poor weight distribution, large working area, and a short period of the sequence. Author generated compact-sized mutually independent PRNGs for the simulation in an extensive distributed system named Twisted PRNG that solves every above-stated problem.

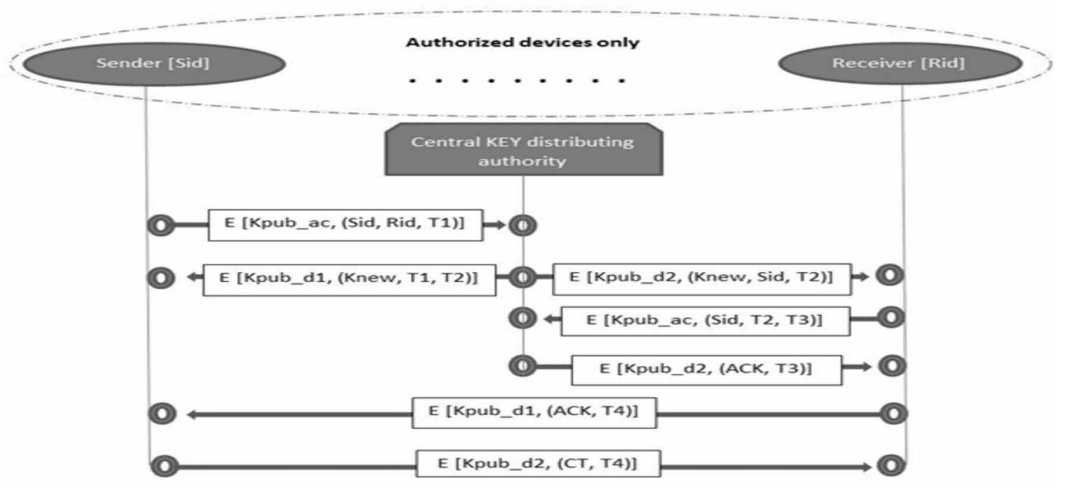
In 1994, the authors presented the Part-2 of the above generator (Matsumoto & Kurita, 1994). The author detected a problem in TGFSR. There was a defect in 'k-distribution' for 'k' more extensive than the order of recurrence. In this study, they came out with a better k-distribution property. They tested the tempering by using weight distribution tests. They divide the generated sequences into pieces conforming binomial distribution and then compare these empirical distributions with the goodness-of-fit test of the hypothesis. They performed the chi-square statistic on the courses grouped in each category. Using Kolmogorov-Smirnov statistics, they measure the difference between their distribution and chi-square distribution.

In 1998, Matsumoto et al. proposed a Mersenne Twister named MT19937 (Matsumoto & Nishimura, 1998). It provides a period of 219937-1, having 623-dimensional equidistribution with the generated sequence of 32-bit length and performs its operation on 624 words of the working area only. The author achieved the computational complexity to be the square of the degree of a polynomial. They used C language for their design and tested their sequences on the Diehard suite. Carrying on his work on Generalized Feedback Shift Registers (GFSR) author introduced the concept of the incomplete array and inversive-decimation method.

Marsaglia, in 2003, termed XOR-shift RNG (Marsaglia, 2003). It consists of consecutive bitwise XOR and shifts operations using seeds. The author claimed that this method could generate high-speed and reliable random bit sequences. The speed of this algorithm family is proven, but some later studies invalidate the reliability claim. The author talked about a simple manipulation (XOR a word with its shifted version). The authors claimed it to produce bits at a speed of 200 million per second. Also, they passed the new diehard battery test except for the binary rank test. Brunt in 2004 has shown some similarity between Marsaglia's XOR-shift RNG and linear feedback shift registers.

Sebastiano Vigna, in 2016, performed some experimental exploration on Marsaglia's XOR-shift generator and proposed XOR-shift* RNG (Vigna, 2016). They replaced the GF_2 operation with a constant value. The generated sequences have periods of 21024-1and 24096-1. It requires only eight logical operations, one addition and one multiplication by a constant. They used the concept of weight to find if a polynomial is dense or sparse.

Figure 1. Communication between authorized devices



Also, in separate work in 2016, only they talked about the XSadd (64-bit shift) generator and pointed out some fault in it (Vigna, 2017). They performed some further scrambling on Marsaglia's XOR-shift generator and proposed their work that covers the weaknesses of XSadd and terms it as XOR-shift128+ and claimed it to pass the strongest BigCrush suite of TestU1. The XOR-shift128+ generates the random sequences of 64 bits. They claimed it to be the fastest full-period generator of that time. They specially mention its uses in the javascript engine of Firefox, Safari, and Google. The authors also talked about XOR-shift128*, XOR-shift1024+, and XOR-shift1024*. This variant, XOR-shift128+, performed with only three-shift, four XORs, and one ADD operation.

In 2017, Umut et al. presented their work on PRNG (Çabuk et al., 2017). They follow Marsaglia's work on XOR shift RNGs. The authors generated three generators. All are abridged versions of XOR-shift+. They made some random scrambling to the original XOR-shift+ and produced a better version named XOR-shiftR+. Authors claimed that their best variant of XOR-shiftR+ is suitable for lightweight applications in terms of randomness and power utilization. They contended that hardware sources of randomness might depend on some environmental conditions so they may be affected, and hence there will be a question mark over their security. They are not much concerned about the space, but of security, and for the same, they reportedly passed every test of randomness.

PROPOSED PUBLIC KEY PROTOCOL

The central 'key' management authority will look after all the generation and distribution of keys to the authorized devices in an IoT environment. Figure 1, communication between authorized devices, illustrates the process.

A sender having its identification as Sid will send a message to authority that he wants to communicate with a device having an ID Rid, Message 1. The admin will then generate a new key, Knew, and will send this new key to the sender and receiver both by using Message 2 and Message 3, respectively. The receiver device will then confirm the Sid from authority by sending Message 4. Authority will acknowledge to the receiver if the Sid received from the receiver node will be the same as sent by an admin previously, Message 5. After receiving the acknowledgment from an authority, the receiver node will send a response to the sender node, Massage 6. After receiving the response from the receiver node, the sender will send its encrypted message, CT, to the sender by using Message 7.

Table 1. Proposed public-key protocol

Message Number	Communicating Devices	Encryption Technique
Message 1	A → S	{A, B, T_1}_pkS
Message 2	S → A	{K_{new}, T_1, T_2}_pkA
Message 3	S → B	{ K_{new}, A, T_2}_pkB
Message 4	B → S	{ A, T_2, T_3}_pkS
Message 5	S → B	{ACK, T_3}_pkB
Message 6	B → A	{ACK, T_4}_pkA
Message 7	A → B	{CT, T_4}_pkB

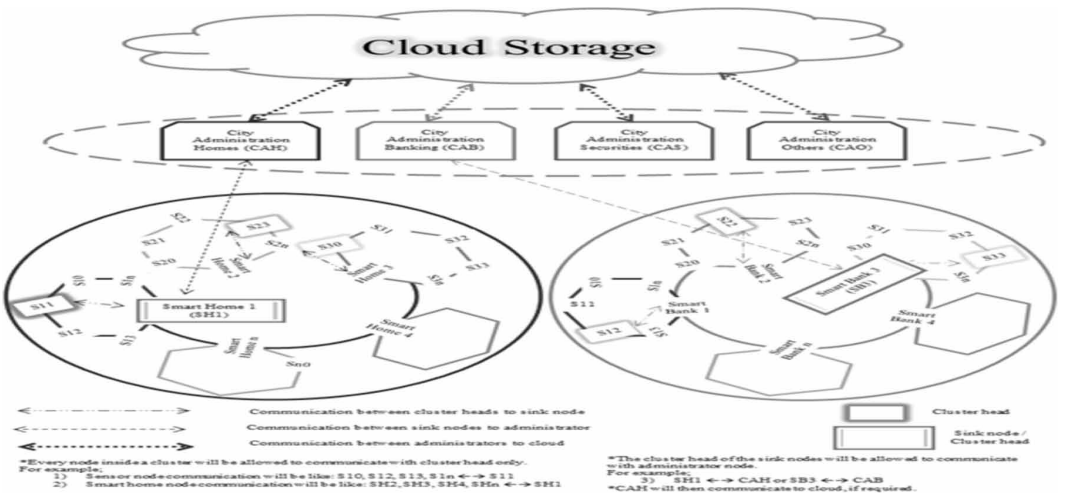
The authorized devices in a cluster can communicate with each other by getting the key from the central 'key' distributing authority. First, the sender node tells the admin about his willingness to talk to the receiver node. Authority then produces separate keys for the sender node and receiver node. The receiver node generates a public key from the received key from authority and sends it to the sender node. The sender node uses this key for further communication.

Table 1 shows the complete process of the proposed public-key protocol. Authors are preventing every message from the adversary by encrypting them with the public key of the intended receiver.

HIERARCHICAL CLUSTERING

Authors use the concept of hierarchical clustering in their said cryptographic environment. In the said smart city environment, the author dedicated different authenticators to different clusters. For example, the city administrator home will take care of the communication between devices that fall under the smart home cluster. The grain level of the hierarchy is the different sensors in a separate home (Elfouly et al., 2017). The regime of communication between devices falls in different groups is shown in figure 2, the hierarchy of communication between devices falls in different clusters. The mechanisms inside a smart home boundary will form one cluster.

Figure 2. The hierarchy of communication between devices falls in different cluster



Every cluster will have one cluster head. Every device inside a group will be allowed to communicate with the cluster head only. The cluster head will then forward the message to the sink node if required. At the sink level, again, there will be a cluster and a cluster head. The cluster head of the sink node will transfer the information to the administrator node on the requirement. This arrangement will save the network from unnecessary flooding. The responsibility of the cluster head will be given to different sensors inside a cluster periodically, keeping in mind the load balancing factor of constraint devices.

PROPOSED METHOD

The cryptographic system is responsible for encrypting a plain text message using a cryptographic key to generate a ciphertext message. The sender encrypts the plain text, and then transmits ciphertext on the channel. The receiver generates the plain text from this ciphertext by performing decryption using the same key used during encryption. In the proposed scheme, the authors unify different modules to get the entire process. The proposed scheme consists of four modules. The first module, named Binary Bit Sequence Generator (BiBiSeG), generates a file of cryptographically secure binary bit sequences containing more than one billion bits. At every identification request it will generate a shuffled instance of that file. The second module, named Random Key Generation (RandKeyGen), creates the public key of variable lengths (128/256/512/1024) for communicating devices (authentication center, a sink node, cluster head, sensors). The KeyConversion module will convert the keys in their free version for mutual authentication. The EncDec modules use the cryptographic key received after mutual authentication for encryption or decryption purposes.

BiBiSeG - Module1

This module is programmed to work in such a way that it takes a very small integer (24/26/28/30/32) as input and generates a very long, more than one billion bits, and cryptographically secure binary bit sequence. This module will be run only for one time at the city administrator side. For different city administrators, programmers can choose distinct integers among 24, 26, 28, 30, and 32. This module also contains a submodule, FileShuffle. Algorithm 1 is the pseudo-code for the BiBiSeG module.

```
Algorithm1: BiBiSeG

Input: Integer among 24, 26, 28, 30, 32
Output: Tested sequence of more than one billion binary bits
d = int 24/26/28/30/32
X = len(d)
Y = Different binary strings of X
for each string
     Z0 = count of zeros
     Z1 = count of ones
     S = Z0 - Z1
if S = = 0
Y= [(Z[p:p+3]) for p in range(0, len(Z), 3)]
for each block
     if [000] & [111] not in Y
return Y
```

The authors obtain different instances of this binary key bit sequences by using the FileShuffle algorithm on the file generated by the BiBiSeG algorithm. Algorithm 2 is the pseudo-code for the FileShuffle module.

```
Algorithm2: FileShuffle

Input: File from BiBiSeG
Output: Shuffled files
f = open(file_path,'r')
data = f.read()
paragraphs = data.split('\n')
random.shuffle(paragraphs)
output = open(output_file_path,'w')
```

Administrators store the files generated by executing this module in its storage. A new file will be provided to the device at every new identification request.

RandKeyGen - Module2

This module randomly selects a binary key bit sequence of the desired length (128/526/512/1024) by using two pointers. Whenever a node requests a key to establish a connection, the authentication center responds to the node with a cryptographically secure key, Knew. The second module will work on the stored files to select a random binary bit sequence each time a new communication is requested.

```
Algorithm3: RandKeyGen

Input: Folder containing ten files
Output: Knew
For i in range[0,10]
     Select rand(i)
For j in range[0,10000000]
     Select rand(j)
d=128/526/512/1024
Knew = string[j+0:j+d]
```

Author termed the proposed key generation processor suitable for the constraint devices because, in the said process, programmers need only two variables for positioning of the curser. Whenever the authentication center is requested, the first variable will choose the file in constant time O(1). Then the second variable will select a starting point in constant time O(1). Overall it will take O(2) for this process to perform its task. Figure 3, proposed key generation method, depicts the working of the proposed key generation method.

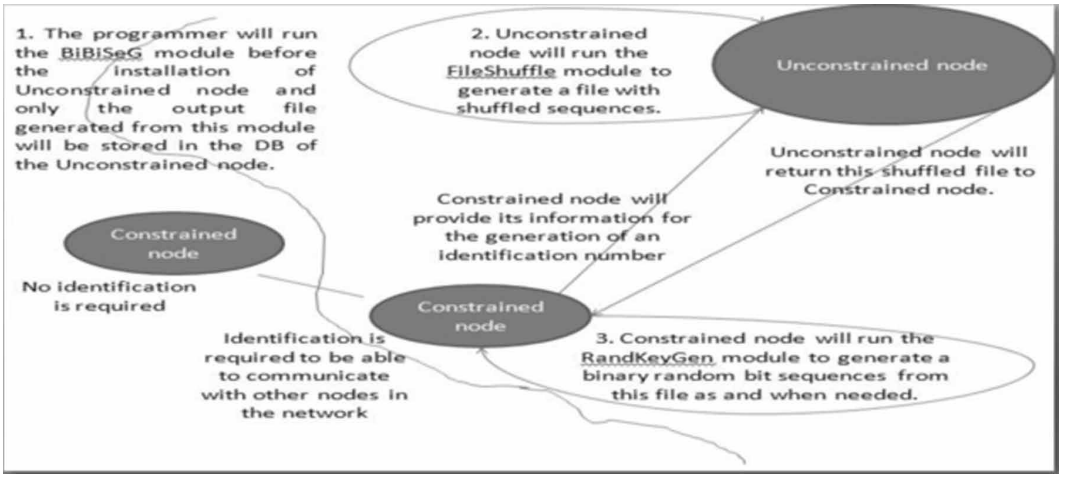
KeyConversion - Module3

The authentication center sends the generated key to the registered devices. Devices can use the public key of the authentication center for further communication. Algorithm 4 shows the process of converting a 128-bit binary key into its free version.

```
Algorithm4: KeyConversion

Input: binary key of the desired length (say 128 bit)
Output: Kpub
d ← 128 bit string
d1 ← first 64 bit string
d2 ← next 64 bit string
d1' ← circular left shift(d1, r)
```

Figure 3. Proposed key generation method




```
d2′ ← circular right shift(d2, r)
d3 ← d2′.d1′
Kpub ← hex(d3)
```

Here ‘r’ can be chosen randomly [1, 64]. The receiver of this key should have to use (r’ = 64-r) in place of ‘r’ to get the exact key for communication.

EncDec - Module4

In this work, the authors are presenting a new cryptographic environment to be used by constraint devices under the IoT environment. Authors are using a local dictionary for converting the sensor’s message into a binary bit sequence. For the encryption, the authors are using the variable (128/256/512/1024) keying mechanism. Whenever the system runs the encryption code, it will get the key of the same length as of plain text. In this module, the authors XORed the plain text with the cryptographic key to get the cipher text. On the receiver end, the receiving node only needs to run the same algorithm using the same cryptographic key for understanding the original plain text. Algorithm 5 shows the process for encryption and decryption.

```
Algorithm5: EncDec

Input: Received text message
Output: Converted text message
RT ← Received text message
RTbin ← (128/256/512/1024) bit long binary sequence
v = length(PTbin)
K1 ← v bit long binary sequence
CTbin = RTbin XOR K1
CTbin ← v bit long binary sequence
CT ← Converted text message
```

IMPLEMENTATION

In the said IoT environment, different clusters will have different dedicated administrators. For example, City Administration Home (CAH) for the group made by Smart Homes, City Administration Banking (CAB) for the clusters made by Smart Banks, etc. Every time a new device enters the group, the administrator first gets it registered. After registration, the device will get some keys for future communications. Section 'A' describes the process for device registration. An authentication server or third party will generate and issue a random nonce for mutual authentication between the devices that want to communicate at a time. Section 'B' describes the process of mutual authentication. Part 'C' presents the means for converting a key into its free version. Every device should follow the protocol presented in section 'D' strictly for secure information transmission.

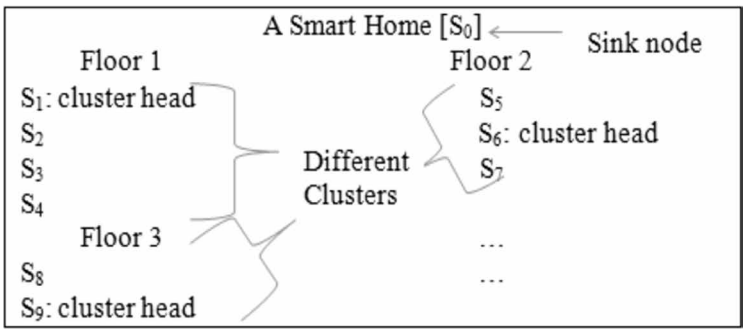
Device Registration

The smart city administrator will keep a record of every sensor under its administration. A sensor will be able to send or receive signals after successful registration only. In a smart city network, every resident and device will have their identification. For example Home ID, School ID, Hospital ID, Bank ID, etc., Resident ID, Doctor ID, Manager ID, Teacher ID, etc., Fan ID, Light ID, Air Conditioner ID, Fridge ID, etc., Patient ID, Shopkeeper ID, Student ID, Customer ID, etc. The smart city administration will give different permissions to different devices at different levels (Guma et al., 2018; Nasution et al., 2018). The capacity of storage, sensing, and transmission will be as per their need. For example, S0, the sink node of a smart home, will have permission to send or receive signals to or from the city administration. S0 will have much permission because it is representing the entire network of a home. S1 is only serving a floor in a smart home network so that it will have lesser capacities, likewise for others. Figure 4, visualization of different sensors in a smart home network, depicts the visualization of different sensors in an intelligent home network.

The smart city administration will make different clusters for different service types. Each group will have its cluster head. Devices inside the cluster will be allowed to communicate with the cluster head only. Remember that machines inside a cluster will have a different dictionary for encryption and decryption purposes.

Also, each cluster will have different methods for encryption and decryption. These methods will add an extra level of security for device communication. Written below is the allowed communication in a group. A considerable number of sensor devices exist in an IoT environment. The concept of the clustering reduces the communication overhead. The authors use the idea of the sink node/cluster head for the same purpose.

Figure 4. Visualization of different sensors in a smart home network



Floor 1 communication will be like:

{S2, S3, S4} → S1
S1 → {S2, S3, S4}

Floor 2 communication will be like:

{S5, S7} → S6
S6 → {S5, S7}

Floor 3 communication will be like:

S8 → S9
S9 → S8

S0, the sink node, is the primary sensor device of this smart home network. Only S0 will be allowed to communicate with the outside world. All the cluster heads from different floors will interact with this node. Below is the communication of a cluster head with the sink node of the environment.

Smart home communication will be like:

{S1, S6, S9} → S0
S0 → {S1, S6, S9}

Each time a new device is detected, it will get registered by the city administrator. After successful registration, the city administration will provide it the public key of authenticating authority [Kpub-ac] along with a device private key [Kdevice] for communication. A new sensor device will register to the City IoT network by providing the details about its Device ID, Cluster name, Environment name. For Example, S2, Floor1, A Smart Home [S0].

[Device ID, Cluster name, Environment name]
New sensor device → Authentication Centre

The authentication center will then provide it with a registration number containing an authentication center's public key and another key that should be used by the device as its private key.

Kpub-ac, Kdevice
Authentication Centre → New sensor device

Mutual Authentication

Devices will then generate their public key by using the received private key and circulate it to the network for future communications by other tools. When a device wants to communicate with another device in the system, it will first ask the authentication center for a new key, Knew, for encryption and decryption.

E(Kpub-ac, (D1, D2, T1))
D1 → Authentication Centre

The device that wants to establish a connection will send its device id along with the receiver's id and a timestamp encrypted with the public key of the mutual authenticator. The authentication center will then generate a new key, Knew, and send it to D1 and D2 both.

E(Kpub-d1, (Knew, T1, T2))
Authentication Centre → D1
E(Kpub-d2, (Knew, Sid, T2))
Authentication Centre → D2

With this, D1 and D2 will get a new key for encryption and decryption purposes. D2 will know that D1 wants to connect.

D2 will now connect with the authentication center to ensure that the received sender id, Sid, is correct.

E(Kpub-ac, (Sid, T2, T3))
D2 → Authentication Centre

The authentication center will match the Sid, and T2 received from D2 with the one sent by the authentication center. The authentication center will acknowledge D2 only after a match; otherwise, it will remain silent.

E(Kpub-d2, (ACK, T3))
Authentication Centre → D2

D2 will then send an acknowledgment message to D1 using the public key of D1.

E(Kpub-d1, (ACK, T4))
D2 → D1

D1 will convert its plain text using Knew and send it to D2 using the public key of D2.

E(Kpub-d2, (CT, T4))
D1 → D2

Upon receiving this CT, the D2 will convert it to PT by using Knew. These processes register the devices in the network, and also they are mutually authenticated before going to initiate any communication. Upon receiving a communication request from any of the 'devices' under the city administrator range, it will execute the 'key' generation process, RandKeyGen module. The following method describes 'key' management and public-key protocol.

EXPERIMENTAL SETUP AND RESULTS

The hardware setup for this work is Intel® Xeon® Silver 4114 CPU @2.20GHZ 2.19 GHz (2 processors) having a 64-bit operating system and x64 based processor on Windows 10 Pro. Spyder (an open-source, cross-platform integrated development environment for scientific programming in the Python language) under Anaconda distribution (a Python data science platform) is used for programming.

Table 2. Obtained CT for the same PT "Acknowledgement" using different key

Key	Ciphertext
Round 1	⅞U⅖⅓y±≈①o⅘X⅙/⅟bc∞j◊
Round 2	j5∞l¥e)y②XG∑M①j>√+□
Round 3	TG~zÖΩ*5B#¹⁄₆$(⅜)>vb□
Round 4	⅝6BF⅜@⅛)EH`!9R⅛"u?◊
Round 5	2%&fc±YQPaμ`Jπ [O⅕□
Round 6	s¹⁄V⅝#:>oG™sw⅝J*¥º □
Round 7	]™vΩ£βOI$Skμ²⁄₅£(zys□
Round 8	<wΩ`≈k0FQTT³9D€¥O,◊
Round 9	?LBL⅞SΩI@L?E{SYi±"□
Round 10	±1{z<¥(m≈LNo³⁄₅C8W⅙π□

The test on the proposed environment is conducted for the same plain text for ten different instances, and every time it produces different ciphertexts. NIST examined the RandKeyGen module for randomness and its uses as a cryptographic key, and found the output sequence generated from the RandKeyGen module suitable to be used as a cryptographic key. Table 2 shows the ciphertext created for each instance using different keys for the same plain text.

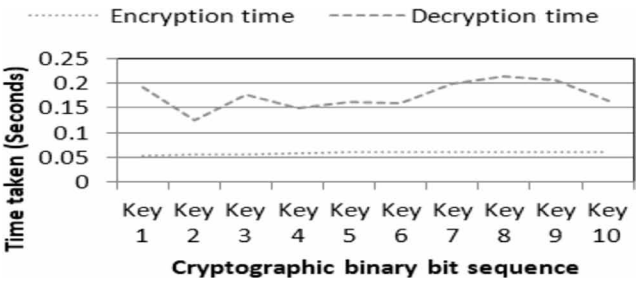
In the said cryptographic environment, variable-length keying is used so that an outsider could not be able to learn the user pattern of the bit sequence. The codes were tested also for the substantial input, and different keys, they noted the time taken by encryption and decryption methods. The example text taken is of 1255 characters. Table 3 gives the obtained values. A graphic presented in figure 5 clearly shows that the time taken for encryption is almost linear, but the time taken for decryption is varying and does not form any linear pattern. Randomly it takes sometimes more time and sometimes less time for decryption. The design of decryption time is twisted.

The environment uses different local dictionaries for different clusters for intra-cluster communication between sensor nodes to the sink node. A node belonging to another group having various local dictionaries will not be able to convert the binary sequence into the correct plain

Table 3. Encryption and decryption time obtained

Key	Encryption Time(Seconds)	Decryption time(Seconds)
Round 1	0.053750	0.191082
Round 2	0.055768	0.124015
Round 3	0.055801	0.178074
Round 4	0.057742	0.149185
Round 5	0.059996	0.162795
Round 6	0.060731	0.160295
Round 7	0.060734	0.199350
Round 8	0.061035	0.213029
Round 9	0.060734	0.206717
Round 10	0.060762	0.165269

Figure 5. Comparison of encryption and decryption times



text if it anyhow manages to get the 'key' bit sequence used to decrypt. Also, different padding on the 'key' bit sequence will be used in different clusters to add an extra level of security. It is assumed for the message length that it is not higher than 18 characters in the said IoT scenario. For the communication between the sink node and cloud service provider, the global dictionary will be used, such as ASCII or UNICODE. The encryption and decryption codes are tested for various input lengths as well as for multiple 'key' inputs. Table 4 shows the obtained result. The first five 'key' rounds convert TEMPERATURE: 30.0C, while the next five rounds of keys convert PRESSURE: 101325Pa.

Table 4. Time taken by encryption and decryption for variable-length input

Proposed cryptographic environment					
Sender node			Receiver node		
Key round	**Ciphertext**	**Encryption time**	**Key round**	**Plain text**	**Decryption time**
1	}M≠+e*:⅞h(NH⅛@H[.h◊	0.046882 second	**1**	TEMPERATURE: 30.0C¸	0.062441 second
2	wh@<}~β~⅔mddm!wP}=□	0.06248 second	**2**	TEMPERATURE: 30.0C¸	0.046851 second
3	AfXb+|¥ H⌐€x¥π<PRZ□	0.062475 second	**3**	TEMPERATURE: 30.0C¸	0.062486 second
4	(ipt⅟€,%N5①yXB∑FQ.◊	0.062487 second	**4**	TEMPERATURE: 30.0C¸	0.062477 second
5	l+Ω3w*9dIr√4j¾ncw⅓□	0.046878 second	**5**	TEMPERATURE: 30.0C¸	0.062476 second
6	j+1}⅜√+y}≈Go①Z+⅝⅜u□	0.060858 second	**6**	PRESSURE: 101325Pa¸	0.062468 second
7	$≥P⅝⅝(6cKIP≡+≈;K92□	0.046879 second	**7**	PRESSURE: 101325Pa¸	0.062489 second
8	?9}③[AKf,G4√WT%①B$◊	0.062478 second	**8**	PRESSURE: 101325Pa¸	0.046869 second
9	<Ih0≤2<cHf②c±CDb∞(□	0.062469 second	**9**	PRESSURE: 101325Pa¸	0.062481 second
10	∞4πI€&⅛84fmw;Snv⅔⅗□	0.062435 second	**10**	PRESSURE: 101325Pa¸	0.062475 second

Table 5. Minimum and Maximum p-value for every test of the NIST test suite (Gupta & Kumar, 2020)

Sr. No.	Name of the Test	p-value (min)	p-value (max)
1	The Frequency (Monobit) Test	0.9991	1.0
2	Frequency Test within a Block	1.0	1.0
3	The Runs Test (128-bits)	0.000047	0.328642
4	Tests for the Longest-Run-of-Ones in a Block (128-bits)	0.023101	0.644213
5	The Binary Matrix Rank Test (38912-bits)	0.039305	0.652234
6	The Discrete Fourier Transform (Spectral) Test (1000-bits)	0.009008	0.561657
7	The Non-overlapping Template Matching Test (1000-bits)	0.435523	0.999519
8	The Overlapping Template Matching Test	2.2101428860723744 e-138	2.2101428860724544 e-138
9	Maurer's "Universal Statistical" Test	0.0	0.0
10	The Linear Complexity Test	0.0	0.0
11	The Serial Test	0.0,0.0	0.0,0.0
12	The Approximate Entropy Test	0.835171	0.999975
13	The Cumulative Sums (Cusums) Test	0.999934143993946, 0.999934143993946	0.9999999999999996, 0.9999999999999996
14	The Random Excursions Test	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
15	The Random Excursions Variant Test	p = None	p = None

SECURITY ANALYSIS

In the proposed lightweight cryptographic environment, authors take care of the security implementation from the very first stage. In the BiBiSeG module, programmers will have a choice to select one input from available five integers (5 options). In the RandKeyGen module, from the selected file containing more than 10000000 (ten million) digits, the program will randomly select one number to start the counting of bits (5 x 10000000 = 50000000 choices). These key bit lengths are also one of the four flavors (128/256/512/1024). So the total number of options will now be (500000000 x 4 = 200000000 (200 million) choices).

The authors tested the keys generated from their proposed PRNG for the suitability of cryptographic uses. The results obtained from the NIST test suite are shown in table 5. From the obtained result, the authors concluded that their proposed method produces cryptographically secure random numbers to be used as key in any cryptographic application.

Before sending its public key, the sender will apply the KeyConversion module. In this module, the sender will have different options for shifting for different key lengths. The shifting will again multiply with the number of choices that come in the previous phase. For a 128 bit key, there will be 63 possible shiftings; for a 256-bit key, there will be 127 possible shiftings; for a 512-bit key, there will be 255 possible shiftings; for 1024 bit key there will be 511 possible shiftings. This will lead to (0.2 billion x 63 = 12.6) billion choices for 128 bit key generation, (0.2 billion x 127 = 25.4) billion choices for 256 bit key generation, (0.2 billion x 255 = 51.0) billion choices for 128 bit key generation, and (0.2 billion x 511 = 102.2) billion choices for 128 bit key generation.

Table 6. Encryption time comparison table

Data Size (kb)	Encryption Time (Parik, Patidar, & Sood, 2003)	Encryption Time (Parik, Patidar, & Sood, 2005)	Encryption Time (Kumar, Kumar, Budhiraja, Das, & Singh, 2016)	Encryption Time (Kumar, Kumar, Budhiraja, Das, & Singh, 2016)	Encryption Time (Proposed)
10	NA	NA	0.071465	0.0329	0.046879
30	0.27	0.27	0.185454	0.0887	0.053750
90	1.03	0.79	0.568465	0.1932	0.059996
240	2.75	2.10	1.835408	0.5226	0.060762

*All times are in seconds.

A brute force attack will take 1,022,000 seconds if it takes one microsecond for implementing one choice to break the secret key. One million twenty-two thousand seconds will lead to 17034 minutes, i.e., 284 full hours, i.e., 12 entire days. So breaking the security of the proposed lightweight cryptographic system by using a brute force attack will have no meaning.

The authors also compared the time taken by their proposed encryption module with the time taken by some other encryption modules and found their module takes less time than other state of the art proposals. The obtained values are presented in table 6.

SECURITY COUNTERMEASURES

According to the reference model of information assurance & security, the programmer should consider every category of information starting from the first stage of the life cycle to ensure the completeness of secure system design. The constant development life cycle consists of security requirements engineering, security design, security countermeasures implementation, security management & monitoring, and secure retirement of an information system. Researchers should prioritize their security goals for proper risk analysis. The researcher presents information taxonomy in four ways: sensitivity, location (controlled, partially controlled, uncontrolled), form (paper, electronic, verbal), and state (creation, processing, storage, transmission, destruction). Programmers should trace the security countermeasures at every stage of the life cycle of a secure system design to preserve the consistency of the system. Security countermeasures are of four forms, namely, organizational (strategy, procedures, audit, governance, policy), technical (cryptography, authentication, authorization), legal (law, contracts, agreements), and human-oriented (training, ethics, culture, motivation, education). Researchers should select the right security countermeasures for cost-effectiveness and efficiency. The security goals can be classified further as integrity, confidentiality, availability, privacy, non-repudiation, audit-ability, authenticity & trustworthiness, and accountability (Ramadan, & Altamimi, 2017).

The presented security scheme is programmed, keeping all the threats in mind. It is replay attack resistance; no device other than the recipient can get the message because the CT (in 7th message) follows four time-stamps T1, T2, T3, and T4. The mechanism does the mutual authentication of the communicating devices before information transmission by using the 6th message. The author makes their cryptographic environment 'server spoofing attack resistance' by providing the public key of the authentication server right at the time of registration of the devices. No one other than the authenticating authority can read message 1. In the proposed cryptographic environment, the authors address the 'no time synchronization problem' by providing different administrators to different local clusters. The authentication server handles the session key generation, and it will be valid for the current communication request only. The session will start from T2 and will end on T4. The proposed cryptographic environment is having resistance from modifies attack as authors use

Table 7. Countermeasures for different threats

Threats	Countermeasures
Replay attack	Message 7 follows T_1, T_2, T_3, T_4
Mutual authentication	Message 6 follows Message 4 and Message 5
Server spoofing attack	Message 1
No time synchronization problem	Local clustering
Session key generation	T_2, T_3, T_4
Modifies attack	Message 7
Local authentication	Local clustering / Edge level
Stolen-verified attack	Message 4
Long persistent attack	Cluster head

the public key of the recipient for every message transmission. The local city administrator provides local authentication to the devices that fall under a specific cluster.

The said environment is using the timestamp as well as the sender and receiver identity to authenticate a communication for a session, so each time a new connection is requested, these attributes will change. A trespasser cannot have any mapping for the current authentication with the previous one. Hence authors claim their cryptographic environment stolen-verified attack resistance. The responsibility of the cluster head will be on rotation. In every cluster, when a node is leaving, or a new node is joining, the cluster head will change. With this arrangement, no device will be left untouched. So the long persistent attack will not be feasible here. Table 7 provides corresponding communication to different countermeasures of the threats.

CONCLUSION AND FUTURE WORK

In this study, the authors proposed a new cryptographic environment suitable for low powered devices of an IoT environment. A random binary bit sequence generator is programmed to generate the cryptographic keys as per the NIST requirement. The use of different shuffled instances of binary bit sequences added an extra level of security in the 'key' generation process. The keys will be used for mutual authentication as well as for encryption and decryption. The concept of hierarchical clustering is used for devices to limit their communication at a local level. Only useful and necessary information will be allowed to transfer to the upper level of the hierarchy. In an IoT environment, devices are large in number, so the programmer uses different authentication centers for load balancing. Authors also presented security analysis and security countermeasures. The result shows that the proposed method is highly suitable for low powered devices of an IoT environment.

Different devices in a cluster will send their data to the cluster head only, and the cluster head will then forward these data to the upper layer if required. In this mechanism, the cluster heads will have an extra burden. So a tool is necessary for the rotation of this cluster head from time to time.

REFERENCES

Auxilia, M., Raja, K., & Kannan, K. (2020). Cloud-Based Access Control Framework for Effective Role Provisioning in Business Application. *International Journal of System Dynamics Applications*, *9*(1), 63–80. doi:10.4018/IJSDA.2020010104

Blum, L. (1986). *Pseudo-random number generator*. Academic Press.

Bussi, K., Dey, D., Biswas, M. K., & Dass, B. K. (2016). Neeva: A Lightweight Hash Function. *IACR Cryptology EPrint Archive, 2016*, 42. Retrieved from http://dblp.uni-trier.de/db/journals/iacr/iacr2016.html#BussiDBD16

Çabuk, U. C., Aydin, Ö., & Dalkiliç, G. (2017). A random number generator for lightweight authentication protocols: Xorshiftr. *Turkish Journal of Electrical Engineering and Computer Sciences*, *25*(6), 4818–4828. doi:10.3906/elk-1703-361

Che, W., Deng, H., Tan, W., & Wang, J. (2008). A random number generator for application in RFID tags. Networked RFID Systems and Lightweight Cryptography: Raising Barriers to Product Counterfeiting, 279–287. doi:10.1007/978-3-540-71641-9_16

Chen, J., Miyaj, A., Sato, H., & Su, C. (2015). Improved lightweight pseudo-random number generators for the low-cost RFID tags. *Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2015, 1*, 17–24. doi:10.1109/Trustcom.2015.352

Chlumecký, M., Buchtele, J., & Richta, K. (2017). Application of random number generators in genetic algorithms to improve rainfall-runoff modelling. *Journal of Hydrology (Amsterdam)*, *553*, 350–355. doi:10.1016/j.jhydrol.2017.08.025

Elfouly, F. H., Ramadan, R. A., Mahmoud, M. I., & Dessouky, M. I. (2017). Efficient Data Reporting in a Multi-Object Tracking Using WSNs. *International Journal of System Dynamics Applications*, *6*(1), 38–57. doi:10.4018/IJSDA.2017010103

Epstein, M., Hars, L., Krasinski, R., Rosner, M., & Zheng, H. (2003). Design and Implementation of a True Random Number Generator Based on Digital Circuit Artifacts. *LNCS*, 2779, 152–165. doi:10.1007/978-3-540-45238-6_13

Gao, L., Ma, M., Shu, Y., & Wei, Y. (2014). An ultralightweight RFID authentication protocol with CRC and permutation. *Journal of Network and Computer Applications*, *41*(1), 37–46. doi:10.1016/j.jnca.2013.10.014

GS1. (2013). *EPC TM Radio-Frequency Identity Protocols Generation-2 UHF RFID*. Specification for RFID Air Interface Protocol for Communications At. 10.1007/s40261-017-0531-2

Guma, I. P., Rwashana, A. S., & Oyo, B. (2018). Food Security Indicators for Subsistence Farmers Sustainability: A System Dynamics Approach. *International Journal of System Dynamics Applications*, *7*(1), 45–64. doi:10.4018/IJSDA.2018010103

Gupta, D. N., & Kumar, R. (2019). Lightweight Cryptography : An IoT Perspective. *International Journal of Innovative Technology and Exploring Engineering*, *8*(8), 700–706. https://www.ijitee.org/download/volume-8-issue-8/

Gupta, D. N., & Kumar, R. (2020). Generating Random Binary Bit Sequences for Secure Communications between Constraint Devices under the IOT Environment. In INCET (pp. 1–6). doi:10.1109/INCET49848.2020.9154009

Gupta, D. N., Kumar, R., & Kumar, A. (2020). Efficient Encryption Techniques for Data Transmission Through the Internet of Things Devices. In V. Jain, O. Kaiwartya, N. Singh, & R. S. Rao (Eds.), IoT and Cloud Computing Advancements in Vehicular Ad-Hoc Networks (pp. 203–228). IGI Global. doi:10.4018/978-1-7998-2570-8.ch011

Kösemen, C., Dalkiliç, G., & Aydin, Ö. (2018). Genetic programming-based pseudorandom number generator for wireless identification and sensing platform. *Turkish Journal of Electrical Engineering and Computer Sciences*, *26*(5), 2500–2511. doi:10.3906/elk-1710-155

Koza, J. R. (1991). Evolving a computer program to generate random numbers using the genetic programming paradigm. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 37–44. Retrieved from http://www.genetic-programming.com/jkpdf/icga1991.pdf

Kumar M., Kumar S., Budhiraja R., Das M. K., & Singh S. (2016). A cryptographic model based on logistic map and a 3-D matrix. *Journal of Information Security and Applications*. 10.1016/j.jisa.2016.09.002

Kumar, M., Kumar, S., Budhiraja, R., Das, M. K., & Singh, S. (2016). Lightweight Data Security Model for IoT Applications: A Dynamic Key Approach. *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData).*

Leonard, P., & Jackson, D. (2015). Efficient evolution of high entropy RNGs using single node genetic programming. *GECCO 2015 - Proceedings of the 2015 Genetic and Evolutionary Computation Conference*, 1071–1078. doi:10.1145/2739480.2754820

Mandal, K., Fan, X., & Gong, G. (2016). Warbler: A Lightweight Pseudorandom Number Generator for EPC C1 Gen2 Passive RFID Tags. *International Journal of RFID Security and Cryptography*, *2*(2), 82–91. doi:10.20533/ijrfidsc.2046.3715.2013.0011

Marsaglia, G. (2003). Xorshift RNGs. *Journal of Statistical Software*, *8*(14), 1–6. doi:10.18637/jss.v008.i14

Matsumoto, M., & Kurita, Y. (1992). Twisted GFSR Generators. *ACM Transactions on Modeling and Computer Simulation*, *2*(3), 179–194. doi:10.1145/146382.146383

Matsumoto, M., & Kurita, Y. (1994). Twisted GFSR Generators II. *ACM Transactions on Modeling and Computer Simulation*, *4*(3), 254–266. doi:10.1145/189443.189445

Matsumoto, M., & Nishimura, T. (1998). Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, *8*(1), 3–30. doi:10.1145/272991.272995

Melia-Segui, J., Garcia-Alfaro, J., & Herrera-Joancomarti, J. (2010). Analysis and improvement of a pseudorandom number generator for EPC Gen2 tags. Lecture Notes in Computer Science, 6054, 34–46. doi:10.1007/978-3-642-14992-4_4

Melià-Seguí, J., Garcia-Alfaro, J., & Herrera-Joancomartí, J. (2011). A practical implementation attack on weak pseudorandom number generator designs for EPC Gen2 tags. *Wireless Personal Communications*, *59*(1), 27–42. doi:10.1007/s11277-010-0187-1

Melià-Seguí, J., Garcia-Alfaro, J., & Herrera-Joancomartí, J. (2013). J3Gen: A PRNG for low-cost passive RFID. *Sensors (Switzerland)*, *13*(3), 3816–3830. doi:10.3390/s130303816 PMID:23519344

Nasution, F. B., Bazin, N. E., Rosalyn, R., & Hasanuddin, H. (2018). Public Policymaking Framework Based on System Dynamics and Big Data. *International Journal of System Dynamics Applications*, *7*(4), 38–53. doi:10.4018/IJSDA.2018100103

Naugle, A. B., Silva, A., & Aamir, M. (2017). Cooperation and Free Riding in Cyber Security Information-Sharing Programs. *International Journal of System Dynamics Applications*, *6*(2), 71–85. doi:10.4018/IJSDA.2017040104

Pareek, N. K., Patidar, V., & Sud, K. (2003). Discrete chaotic cryptography using external key. *Physics Letters. [Part A]*, *309*(1), 75–82. doi:10.1016/S0375-9601(03)00122-1

Pareek, N. K., Patidar, V., & Sud, K. (2005). Cryptography using multiple one dimensional chaotic maps. *Communications in Nonlinear Science and Numerical Simulation*, *10*(7), 715–723. doi:10.1016/j.cnsns.2004.03.006

Peris-Lopez, P., Hernandez-Castro, J. C., Estevez-Tapiador, J. M., & Ribagorda, A. (2009). LAMED - A PRNG for EPC Class-1 Generation-2 RFID specification. *Computer Standards & Interfaces*, *31*(1), 88–97. doi:10.1016/j.csi.2007.11.013

Picek, S., Jakobovic, D., Knezevic, K., & Derek, A. (2019). C3PO: Cipher construction with cartesian genetic programming. *GECCO 2019 Companion - Proceedings of the 2019 Genetic and Evolutionary Computation Conference Companion*, 1625–1633. doi:10.1145/3319619.3326869

Poorghanad, A., Sadr, A., & Kashanipour, A. (2008). Generating high quality Pseudo Random Number using evolutionary methods. *Proceedings - 2008 International Conference on Computational Intelligence and Security, CIS 2008, 1*, 331–335. doi:10.1109/CIS.2008.220

Ramadan, R. A., & Altamimi, A. B. (2017). Hierarchal Fuzzy Logic Controller and Internet of Things (IoT) Information: Disease Spreading as a Test Case. *International Journal of System Dynamics Applications*, *6*(3), 59–86. doi:10.4018/IJSDA.2017070104

Rukhin, A., Soto, J., & Nechvatal, J. (2010). SP800-22rev1a, (April), 131.

Salustowicz, R., & Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, *5*(2), 123–141. doi:10.1162/evco.1997.5.2.123 PMID:10021756

Stipcevic, M., & Rogina, B. M. (2006). *Quantum random number generator, 7*. Retrieved from https://arxiv.org/abs/quant-ph/0609043

Tomassini, M., Sipper, M., Zolla, M., & Perrenoud, M. (1999). Generating high-quality random numbers in parallel by cellular automata. *Future Generation Computer Systems*, *16*(2), 291–305. doi:10.1016/S0167-739X(99)00053-9

Vasyltsov, I., Hambardzumyan, E., Kim, Y. S., & Karpinskyy, B. (2008). Fast digital TRNG based on metastable ring oscillator. Lecture Notes in Computer Science, 5154, 164–180. doi:10.1007/978-3-540-85053-3_11

Vigna, S. (2016). An experimental exploration of Marsaglia's xorshift Generators, Scrambled. *ACM Transactions on Mathematical Software*, *42*(4), 1–23. Advance online publication. doi:10.1145/2845077

Vigna, S. (2017). Further scramblings of Marsaglia's xorshift generators. *Journal of Computational and Applied Mathematics*, *315*, 175–181. doi:10.1016/j.cam.2016.11.006

Wu, J., & O'Neill, M. (2010). Ultra-lightweight true random number generators. *Electronics Letters*, *46*(14), 988. doi:10.1049/el.2010.0893

Deena Nath Gupta is a Research Scholar in the Department of Computer Science, Faculty of Natural Sciences, Jamia Millia Islamia (Central University), New Delhi-110025, INDIA. He did his B. Tech. from ABES Engineering College, Ghaziabad-201009, Uttar Pradesh, India, and M. Tech. from Galgotias College of Engineering and Technology, Greater Noida, G. B. Nagar-201310, Uttar Pradesh, India. He has an excellent academic background with a very sound educational and research experience. He has published various research papers in the conferences of international/national repute. His research interests include Cyber Security, Sensor Security, Network Security, Lightweight Cryptography, etc.

Rajendra Kumar is presently working as Professor in the Department of Computer Science, Faculty of Natural Sciences, Jamia Millia Islamia (Central University), New Delhi-110025, INDIA. He has an excellent academic background with a very sound educational and research experience. He has published various research papers in the Journals and conferences of international/national repute. His research interests include Cyber Security, Cloud Security and Privacy, Big Data Analytics, Data Mining, IoT, Software Security, Requirements Engineering, Security Policies and Standards, Software Engineering, Access control, and Identity Management, Vulnerability Assessment, etc.