An Improvised Algorithm for a Dynamic Key Generation Model

D. V. Guru Saran and Kurunandan Jain

Abstract Data may be kept private, secure, and authentic via cryptography. Computational resources and communication channel performance impede the development of an efficient key generation model in IoT devices for encryption and decryption applications. Most IoT networks encode information with session keys, which are less secure than dynamic key encryption used in communication channels. Furthermore, because of the resource limitations of most IoT networks, it is impractical to use the current dynamic key generation approaches. To solve these problems, a dynamic key generator model has been designed that continuously generates unique keys in the range of 1000–10,000,000. We also discuss our proposed model's security and performance analysis to validate the feasibility of its operation in such a resource-constrained IoT environment.

Keywords IoT · Dynamic key · Cryptography · Security

1 Introduction

It is only a matter of time until the Internet of Things (IoT) becomes an increasingly crucial part of our everyday lives, thanks to the arrival of 5G technology. A new security danger is created, or malicious attackers can get into the information and equipment that are supposed to be secured [1]. Certificate-based encryption and public key distribution are adequate to assure data confidentiality, integrity, and validity at OSI network layer levels. These services are unsuitable for IoT devices due to their memory, computational power, energy consumption, and equipment space limitations [2]. In addition to standard network security weaknesses, the Internet of Things offers new security challenges due to its unique features and the fact that it

D. V. G. Saran (✉) · K. Jain
Center for Cyber Security Systems and Networks, Amrita School of Engineering, Amritapuri Campus, Vallikavu, Kerala, India
e-mail: amenp2csn20010@am.students.amrita.edu

K. Jain
e-mail: kurunandanj@am.amrita.edu

S. Smys et al. (eds.), *Inventive Computation and Information Technologies*, Lecture Notes in Networks and Systems 563, https://doi.org/10.1007/978-981-19-7402-1_43

includes several sensor nodes. Data transiting through the Internet of Things (IoT) network is protected by lightweight cryptographic algorithms [3].

Data confidentiality, integrity, and authenticity may all be ensured by using cryptography. The information must be kept private to prevent it from being accessed or misused by others. Authenticity ensures that data cannot be disputed, whereas data integrity ensures that it is trustworthy and reliable information being carried across the network [4].

Cryptographic keys are used for encryption and decryption to ensure security. These keys encrypt the data by converting plaintext to ciphertext. A cryptographic key can be created in one of two ways. Symmetric and asymmetric keys are two sorts of keys. In symmetric key encryption, the very same secret is used to encode and decode the message. In asymmetric keys, one encrypts the message with one key and decodes it with another key [1, 5]. While dynamic keys are used once for each message transmitted, session keys are used to encrypt and decrypt the information in a session to ensure session security. The system's security is enhanced because dynamic keys are utilized for every data transmission [6].

In the IoT network context, malicious users can target the data or devices that generate the data. Active and passive attacks on an IoT network/system are two types of attacks. Passive attacks include the attacker listening to and analyzing traffic passed between objects without altering it. This enables the attacker to ascertain sensitive data like credentials or keys shared, which can be abused to perform cyberattacks. During cyberattacks, the attacker alters, deletes, or replaces transmitted data with malicious messages [7]. It can also impersonate a legitimate node, engage in the key formation procedure, or replay valid messages to gain the system's trust to steal sensitive data such as cryptographic keys.

In previous research [8], we created and implemented a method for generating discrete, transitory keys known as dynamic keys, which are used to determine the delivery time of numerous packets in data transmission or the delivery time of a single packet. Using symmetric key encryption, the suggested dynamic key generation mechanism is developed to work in IoT environmental system. In this investigation, we assume that every individual message transmission between communicative parties uses the dynamic key created by our technique. In comparison with conventional dynamic key generation approaches, we provide a robust system for cryptanalytic assaults while using fewer resources. Three steps comprise the key generation algorithm of the cyclic dynamic key generator (CDKG). It utilizes an input pair of secret and seed to create dynamic keys without a key distribution center (KDC) intervention.

This paper redesigns the algorithm to boost efficiency while preserving strong resilience to cryptanalytic attacks. Furthermore, we replace the SHA256 hash with the Blake2b hash because BLAKE2 is much more efficient in software over most platforms than SHA256. When hashing 64 bytes, for example, it is often twice as efficient as SHA256 on ARM devices [9]. BLAKE2 includes potentially valuable security features such as protection against length extension, a common technique of keying, "personalization tags" to ensure domain separation, etc. In addition, we implemented configuration stage and random stage, which will be discussed in Sect. 3.

The following are the main contributions for this paper:

1. Redesign and enhance the model's performance for dynamic key generation intended for use in an IoT ecosystem.
2. Analyze the efficiency and safety of the suggested method for protecting IoT devices.

2 Literature Survey

Most key management solutions prefer to encrypt and decode messages with session keys. Using session keys to encrypt data is vulnerable to eavesdropping or compromising the data. Using a session key for more than one session may compromise the data, so dynamic keys are used because they are one-time use keys employed per message instead of per session. Dynamic keys are preferred over session keys due to their security resilience [6], whereas session keys are preferred for computation and storage purposes.

In this research, Thanh Nha Dang [10] suggested an improved AES scheme that creates dynamic keys. A 16-byte data packet was delivered in a numerically indexed pattern. Based on the pattern, the key in an Internet of Things (IoT) system was dynamically updated with encrypted data.

At the PHY layer, the researchers researched, assessed, and uncovered flaws in OFDM-based encryption methods [11]. It is been discovered that frequency response encryption reduces the effects of channel fading and improves bit rate error performance. A new method for altering encryption blocks for input OFDM frames and a dynamic secret key mechanism has been presented to strengthen the security of OFDM-based encryption systems.

Uchiteleva's method dynamically updated the secret key on both endpoints of a network connection by using pseudo-random (PR) [12] sequences produced at the physical layer of wireless transmitters throughout the data transmission. In an Industrial Internet of Things (IIoT) setting, this method is adaptable and suited for large network nodes with limited computational power.

Utilizing a Diophantine variant of the nonlinear equation [13], Thirumalai's suggested solution repurposes the RSA technique for storage efficiency. In addition, this technique worked exceedingly well due to its sole use of an RSA public key. MEMK does not involve a multiplicative inverse function or extended Euclid's method. To acquire a test result for MEMK PKC key generation, encryption, and decryption phases, they ultimately increased the N-bit modulo bits from 1 to 10 K.

Moosavi [14] provided two ECG-based cryptographic key generation algorithms. They first suggested using the Fibonacci linear feedback shift register (LFSR) pseudo-random number generator to generate a sequential set of APIs. Second, they demonstrated a key generation mechanism that is both highly secure and low cost. Researchers discovered that traditional IPI-based solutions take 12.3% and 41.2% more time when comparing key generation execution times.

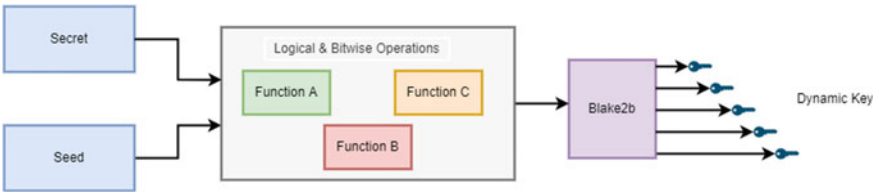

Fig. 1 General overview of key generation methodology

This research proposed the generalized triangle-based security algorithm [15] as an energy source, resource data encryption algorithm with an acceptable encryption generation mechanism (G-TBSA). The proposed G-TBSA is implemented in constrained Wi-Fi wireless sensor networks (WSNs). The key generation process is at the algorithm's core because it requires low resources to produce keys, reducing algorithm complexity and improving energy efficiency.

3 Proposed System

3.1 General Overview of the Key Generation System

Figure 1 depicts a broad overview of the key generation approach. The three functions use logical and bitwise operations to produce random keys supplied as an input to the Blake2b cryptographic hash function. Using the Blake2b hashing technique, a 256-bit hash is generated. The result generates the dynamic key from the specified secret and salt. There are four stages of operation in the key generation scheme: the initialization stage, the configuration stage, the randomness stage, and the cyclic stage.

3.2 Initialization Stage

Figure 2 depicts the broad operational perspective of the initialization stage. Two pre-shared alphanumeric values are required as inputs for this stage: a secret and a seed. Only Function A, the configuration stage, and BLAKE are active at this level. The configuration stage is contained inside Function A. After completing the configuration stage, it executes logical and bitwise operations to generate a new secret. The freshly generated secret is sent to Function C or Function B along with the Blake2b hashing method. Blake2b combines the pre-shared salt and freshly produced secret to construct a 256-bit dynamic key. The key generation algorithm then moves on to the cyclic and random stages.

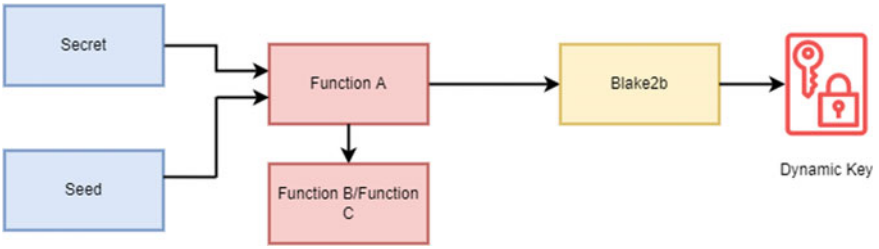

Fig. 2 Initialization stage

3.3 Configuration Stage

This stage is implemented in Function A. Instead of sending the input directly to logical and bitwise operations, it performs mathematical operations on the secret and seed pair, as shown in Fig. 3. Two pre-shared alphanumeric values are inputs in the configuration stage, i.e., a secret and seed, combined and stored in the input. The combined input is divided into six equal groups by padding with zero. Apply mod 6 to these six equal groups and make adjustments such that S1, S2, S3, S4, S5, and S6 are obtained. After the configuration stage, the divided input performs logical and bitwise operations in Function A, yielding a new secret as an output. The newly created secret is sent into Function C/Function B and the Blake hash algorithm.

3.4 Randomness Stage

The general workings of the randomness stage are indicated in Fig. 4. This stage is implemented in Function C. The stage requires a secret generated from Function A and a key generated from the previous result, i.e., a dynamic key. Function C performs logical and bitwise operations on the secret generated from Function A and the key generated from the previous result to generate a new secret. The new secret is supplied to function B as an input. If there is no previous result stored in memory, the new secret generated from Function A is directly passed to Function B. By implementing this stage, for the same secret and key, we generate different sets of random keys.

3.5 Cyclic Stage

Figure 5 depicts the main functioning overview of the cyclic stage. At this level, all modules are operational (Function A, Function C, Function B, and BLAKE). Function A aims to create new secrets from Function B's output pairs. Function C's

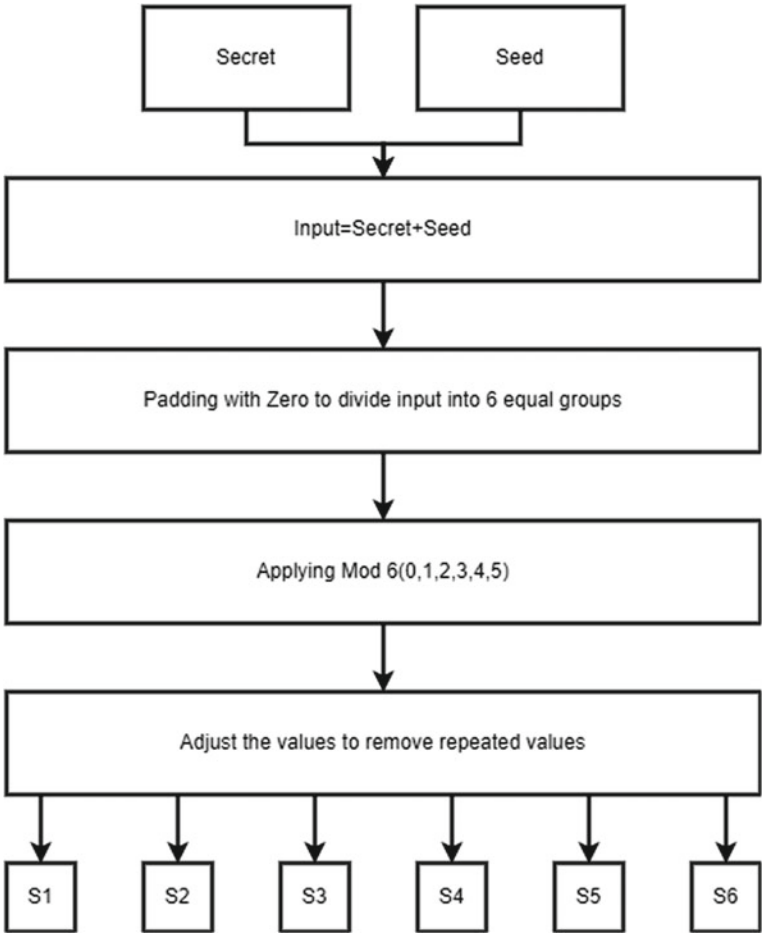

Fig. 3 Configuration stage

purpose is to give Function B a secret. From the outputs of Function A and Function C, Function B tries to create fresh secrets and seeds for Function A. The secrets created by Functions A and B are continuously fed into the 256-bit dynamic key generating Blake hash function.

4 Key Generation Methodology

This section provides a detailed summary of each function. Table 1 gives each function's terms and notation used in this section.

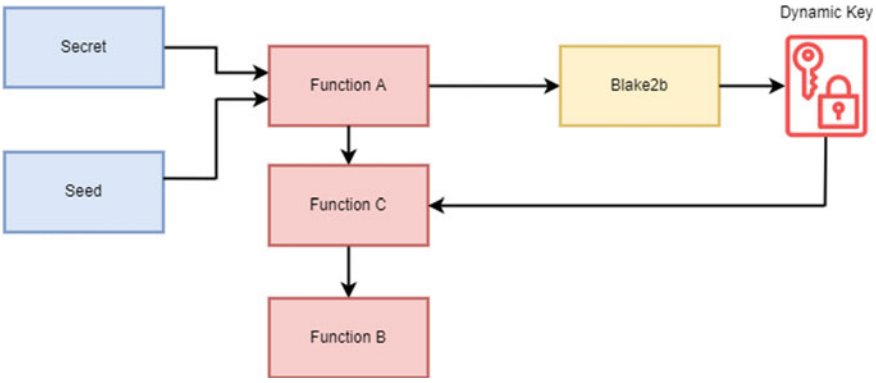

Fig. 4 Randomness stage

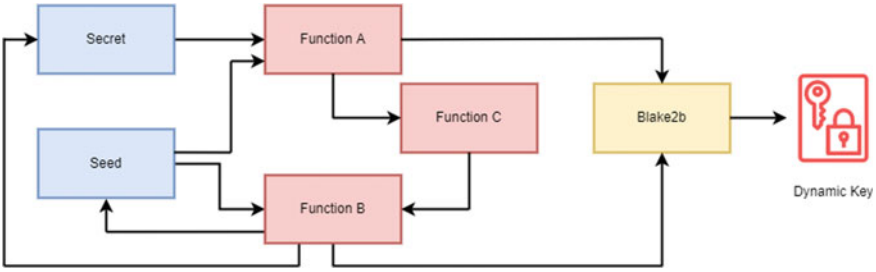

Fig. 5 Cyclic stage

Table 1 Terms and notations

Symbol	Notation
XOR	Bitwise exclusive
LCS	Left circular shift
RCS	Right circular shift
LRS	Logical right shift
LLS	Logical left shift

4.1 Function A

Figure 6 depicts the inner workings of Function A in detail. The secret and seed are first transferred to the configuration stage. The detailed working of the configuration stage is indicated in Fig. 3. After the configuration stage, the input divides the secret into six parts: S1, S2, S3, S4, S5, and S6; S5 and S6 undergo the left and right circular shift operations, respectively. The circularly shifted S5 and S6 perform an Exclusive OR operation with S3 and S4, resulting in S7 and S8. Similarly, S3 and S4 perform the left and right circular shift operations, respectively. The circularly shifted S3 and

S4 perform an Exclusive OR operation with S1 and S2, yielding S9 and S10. S7, S8, S9, and S10 are subjected to a logical right shift operation, with the resulting outputs combined together to generate a new secret.

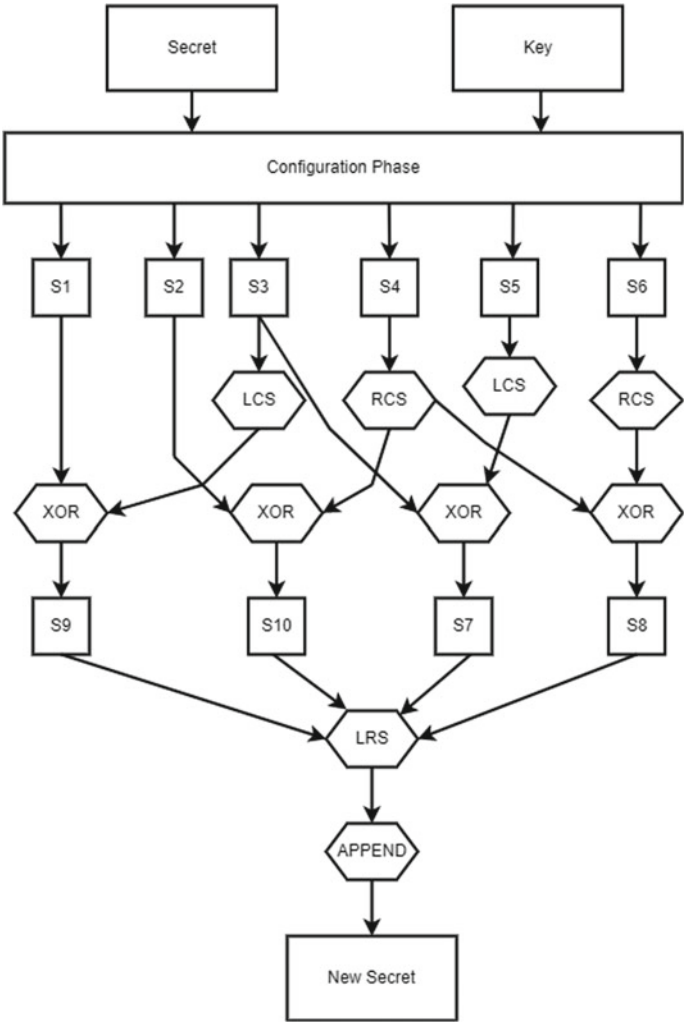

Fig. 6 Function A

4.2 Function C

Figure 7 illustrates the inner workings of Function A in great detail. The stage requires a secret produced by Function A and a key produced by the preceding result. This function C is valid if it returns the previous value. If there is no prior result, Function A immediately passes the newly produced secret to Function B. The method divides the secret and the key into four parts: S'1, S'2, S'3, and S'4. K'1 and K'3 suffer circular shifts to the left and right, respectively. Circularly shifted K'1 and K'3 are subjected to an Exclusive OR operation, with S'1 and S'3 producing S'6 and S'5. S'2 and S'4 suffer circular shifts to the left and right, respectively. S'8 and S'7 are created when S'2 and S'4 are circularly moved. The outputs of the logical right shift operation performed on S'5, S'6, S'7, and S'8 are concatenated to produce a new secret.

4.3 Function B

It is shown in Fig. 8 how Function B works from a high-level perspective. S"1 to S"4 are the four components of the secret that the algorithm divides into and the seed into two parts (K1 and K2). Shifts in the direction of left and right circular motion occur for K1 and K2. As a result of this process, K4 is generated. A left circularly shifted K1 conducts an Exclusive OR operation with S"3 to create K3 as a consequence of the Exclusive OR operation with S"1. K3 and the result of S"2's left circular shift is supplied as input to an Exclusive OR operation, which generates the output. S"6. Using Exclusive OR, the result of the right circular shift operation on S"1 and K4 is used to produce the desired output. S"5. Like S"3, the result of Exclusive ORing S"2 and a right circularly shifted S"3 is produced by S"3. S"7. An Exclusive OR operation on S'1 with a left circular shift is carried out S"4 is responsible for the final product. S"8. After performing a logical left shift operation on each S5, S6, and S7, the outputs are appended together to form a new secret. Logical right shift is applied to K3 and K4, and the outputs are appended together to form a new seed.

5 Security Methodology

This section goes through the numerous attack vectors used against the proposed approach key system. These tests demonstrate that the proposed approach key system is resistant to different attacks that an attacker might attempt to reveal the keys.

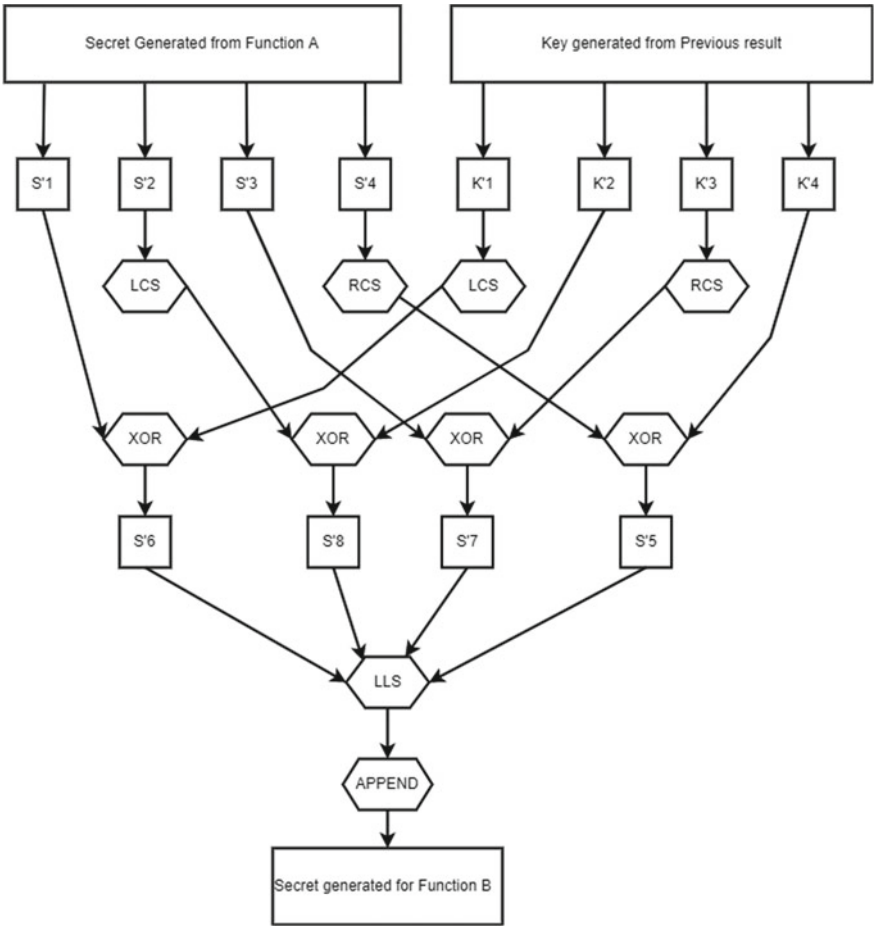

Fig. 7 Function C

5.1 Chosen Plaintext Attack

A cryptanalysis attack can result in illegitimate access to sensitive data due to the compromised symmetric keys. The adversary sends plaintexts of his or her choosing to the encryption oracle to get comparable cipher messages. The adversary attempts to deduce the keys used by the oracle based on the plaintext–ciphertext pairs retrieved. Figure 9 depicts the general attack scenario. Due to the exposure of symmetric keys, this attack may result in unauthorized access to sensitive data.

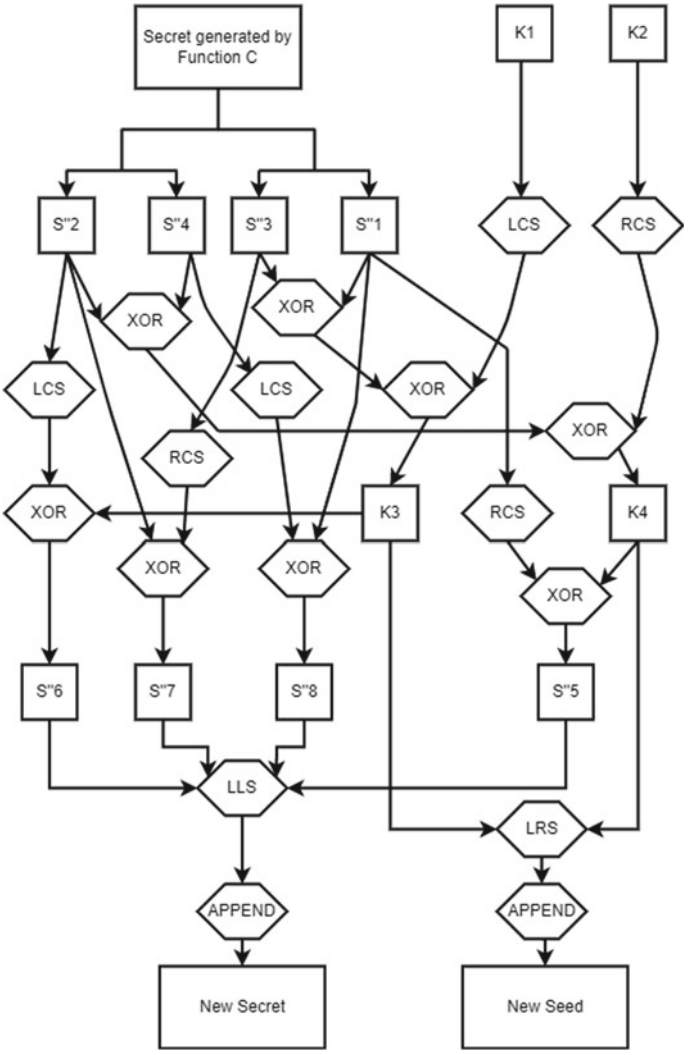

Fig. 8 Function B

Fig. 9 Chosen plaintext attack

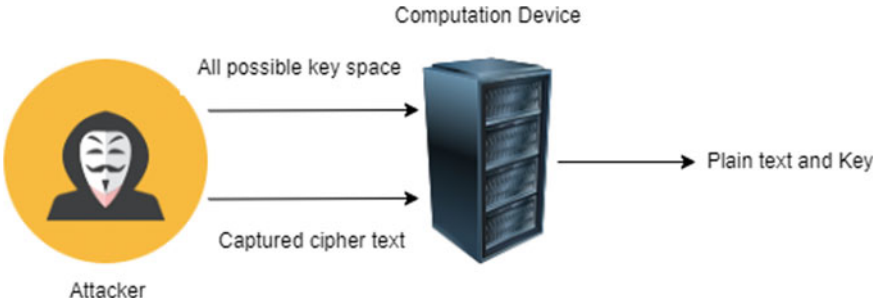

Fig. 10 Brute force attack

5.2 Brute Force Attack

The term "exhaustive key search methodology" refers to this cryptanalysis method. In this attack, the decryption of ciphertext using all possible key spaces determines the key used to encrypt the data. The strength of the algorithm's cryptographic keys may be evaluated using this attack. The key to one message on the transmission medium can be used to decode all other messages if the attacker is successful. As seen in Fig. 10, the general assault scenario. An attacker cannot use the same key to decode many messages in the same transmission medium because of the dynamic key. There must be a way for an attacker to brute force an initial seed or secret, in addition to being able to authenticate message decryption using the dynamic key generation method.

5.3 Preimage Attack

The attack puts the hashing technique to the test against preimage resistance. This attack vector has two variants: the first and second preimage attacks. A first preimage attack allows the attacker to identify a message with a hash length of less than 2^N. Resistance to the first preimage assault indicates that it is impossible to identify any second input, i.e., given message M. It is impossible to create a second preimage M' $\neq M$ such that $H(M) = H(M')$.

5.4 Replay Attack

This is a version of the Man-In-The-Middle (MITM) attack, in which attackers capture network communication packets and retransmit tampered but genuine packets to impersonate the legitimate user. This form of attack is frequent when

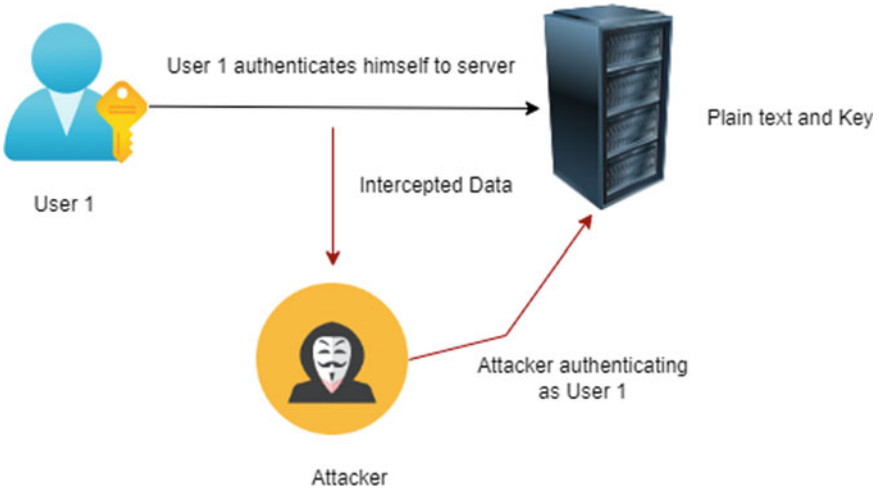

Fig. 11 Replay attack

a legitimate system uses reusable authentication keys. This attack may allow an adversary to gain access to system resources. Attack scenarios are depicted in Fig. 11.

5.5 Session Hijacking

After acquiring the stolen session key or token, an adversary operating as a gateway can get unauthorized access to sensitive information and applications in devices/services/networks. This attack can be carried out by either stealing or guessing an authentic session key to acquire unauthorized access to the system. Figure 12 depicts the general attack scenario. This attack methodology can aid in determining whether a suggested scheme for session monitoring is possible.

6 Performance Analysis

The suggested dynamic key scheme's performance measures are outlined in this section. Randomness, storage, and computation requirements assess if the algorithm can work in an IoT environment.

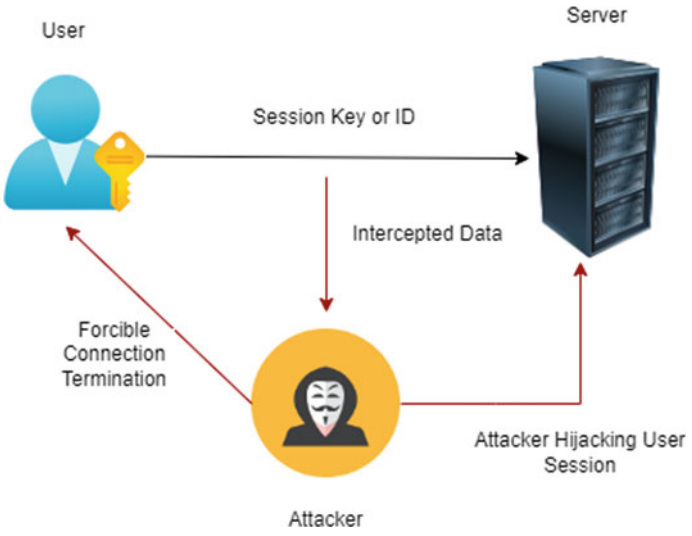

Fig. 12 Session hijacking attack

6.1 Test Configuration

The key generation scheme was tested on the Raspberry Pi 4 Model B, a tiny single-board device commonly used in IoT networks [16, 17]. Table 2 lists the hardware specifications for this single-board computer.

Before transmitting data across the network, this single-board computer may process raw data obtained from different sensor feeds and execute cryptographic procedures. The suggested approach is intended to produce dynamic keys spanning from 1000 to 10,000,000, and various performance features in compute, and storage requirements are tested. The prototype was built using Python 3, an interpreted, high-level, general-purpose programming language. All Python libraries required for the prototype's proper operation are supported by the Raspberry Pi OS, which runs on the hardware.

Table 2 Raspberry Pi 4 model B hardware specifications

Version	Type
Instruction set	ARMv8 (64/32-bit)
SoC	Broadcom BCM2711
FPU	Neon-FP-ARMv8
CPU	4 X Cortex-A72 @ 1.5 GHz
GPU	Broadcom VideoCore VI 500 MHz
Memory (SDRAM)	1 GB

6.2 Randomness Analysis

Randomness analysis assures that the output created by an algorithm is pattern less. This analysis helps to confirm that the produced keys are unique. The National Institute of Standards and Technology (NIST) proposed several to verify the randomness of generated dynamic keys [18, 19]. These tests use conventional normal and chi-square (χ 2) distributions as reference distributions. If the pattern under analysis is not random, the resulting test statistic will be in the extremes of the reference distribution. Table 3 provides an overview of the test completed.

7 Results and Discussions

This section describes the proposed dynamic key generation scheme's security features and the results of several testing.

7.1 Security Features

1. Mutual Authentication: Mutual authentication happens when both sides of a communications connection, rather than simply one, authenticate one others identity. Most IoT devices require a connection to a remote server to work correctly. They may also be required to connect to other IoT devices. IoT devices must communicate via an untrustworthy network (the Internet). Mutual authentication decreases the possibility of attackers compromising their connections by guaranteeing that the data they receive is trustworthy and from a legitimate source.
2. Session keys security: In each communication session, dynamic keys are limited to a fixed amount of messages ranging from one to hundreds. On the other hand, session keys are utilized for the duration of the communication session. The maximum number of messages that can be encrypted with a single dynamic key exceeds the maximum number of messages that may be exchanged in a communication session.
3. Perfect forward secrecy: Forward secrecy (FS) is a cryptographic primitive that changes the keys used to encrypt and decrypt data automatically and frequently. Even if the most recent key is compromised, just a small amount of necessary data is released due to this ongoing process. Even if the server's private key is compromised, absolute forward secrecy helps protect session keys. Since dynamic keys could only be used once in messages in a single communication session, they add an extra layer of protection to secrecy.
4. User Anonymity: Intruders continuously look for security weaknesses and sensitive data such as passwords and user IDs. Intruders can use these credentials

Table 3 NIST test overview

Name of the test	Description
Frequency test	There should be an approximately equal amount of ones and zeros in a series, which is what the test measures
Frequency test within a block	Ones in an M-bit block are being tested to check if their frequency equals M/2
Run test	The number of one- and zero-runs of varying durations is tested to see if the random distribution using the runs test. With this test, you can see if the rate of change between 0 and 1 s is too rapid or too sluggish
Test for the longest run of ones in a block	Tests if the longest run of ones in the tested sequence is consistent with the longest run anticipated in a random distribution
Binary matrix rank test	Tests if fixed length subsequences of the original sequence are correlated linearly
Discrete Fourier transform test	Assuming randomness is the assumption, this approach checks the analyzed sequence for periodic characteristics (i.e., recurrent sequences that are near together). We are looking for a noticeable difference between 95 and 5% in peak counts, surpassing that requirement
Non-overlapping template Matching test	With this test, we are trying to determine which generators create an abnormally high number of non-periodic (aperiodic) patterns we have defined
Overlapping template matching test	A pre-specified objective string's frequency is examined using the overlapping template matching test. To distinguish this from the non-overlapping matching test, the window only advances one bit before the search is resumed when the pattern is found. This is the sole difference between the two
Maurer's universal statistical test	With this test, we are looking to see whether there is any way to reduce sequence size without sacrificing quality dramatically. It is possible to reduce the size of a non-random sequence considerably
Linear complexity test	Determines if the sequence is sufficiently complicated to be considered random. Longer LFSRs can distinguish random sequences. An LFSR that is too short shows non-randomness
Serial test	We will examine if the 2 m m-bit overlapping patterns appear as frequently as would be anticipated from a random sequence in this test

(continued)

Table 3 (continued)

Name of the test	Description
Approximate entropy test	The topic of this test is the frequency of all conceivable overlapping m-bit patterns in the whole sequence. With this test, we are looking to see how often blocks of lengths m and m + 1 overlap with the predicted frequency of random blocks
Cumulative sums test	In order to identify if the examined sequence contains too many or too few component sequences, a test is conducted. The overall cost is about the same as going on a walk at random. For some non-random sequences, the random process's deviations from zero will be considerable
Random excursion test	This test aims to examine if the frequency of a certain state's occurrence in a cycle differs from the random sequence. There are eight tests in this assessment, one for each of the states: −4, −3, −2, −1 and + 1, + 2, + 3, + 4. There are eight tests in this evaluation (and conclusions)
Random excursion variant test	This experiment aims to see if the unpredictability of the number of trips to different states differs from what is predicted. Each state −9, −8,…, −1 and + 1, + 2,…, + 9 is represented by one of the eighteen tests included in this study (and the conclusions drawn from them)

to carry out MITM and impersonation attacks. We can keep specific network identities hidden when exchanging messages using dynamic keys.

5. Key dynamics: When you utilize a fixed key for an extended period, the likelihood of breaking it increases. This method generates a new key each time it starts, recognizing the dynamics of the key and significantly increasing its security.
6. Full randomness test pass: The redesigned model passed all the series of randomness tests given by NIST

7.2 Security Test

1. Cryptanalysis and Brute force attacks: The algorithm generates a one-time master key that is used to secure individual messages rather than entire sessions. It functions similarly to a one-time pad. The attacker never finds the keys using the ciphertext attack because analyzing encrypted messages with many unique keys is difficult. The breach of a single key reveals only one message; it is impervious to cryptanalysis assaults. A critical space of 1.1579209e + 77 is required to brute force a 256-bit key. In theory, it would take 3.671052 years for a supercomputer with a speed of 100 petaflops to deplete all 256-bit key space. Rendering

this type of attack against the redesigned key generation mechanism is infeasible. However, this situation is only applicable to fetching a single dynamic key. Because the key space is the same size, the amount of computing required to brute force the input seed and secret pair and validate the decryption of individual information increases nearly 10 million times to decipher a complete communication session correctly.

2. Replay and Session Hijack Attacks: Replay attacks are common on network devices that use reused authentication keys. By employing the provided dynamic approach, the attacker is prevented from using the same access code to get access to network resources. To prevent replay attacks, a single dynamic key can be used to generate an access code that can only be verified once. If an attacker obtains the session key, he or she can take over a user session by intercepting the user's session key and impersonating this user to maintain the connection with the system. This attack strategy is not conceivable in our redesigned model because session communications are encrypted using dynamic keys.

8 Hardware Consumption

The algorithm is efficient regarding entropy and storage because the prototype and keys take up only 7 KB of code and key storage space, as given in Table 4. Ten million keys were produced from an alphanumeric seed and a secret during the prototype stage. Figure 13 depicts the rise in time as the number of keys, increases, and we also compared the redesigned algorithm to the previous algorithm. The process takes 4688 s to produce ten million keys. While creating ten million dynamic keys, the technique consistently used less than 20 MB of RAM.

Table 5 gives the results of the NIST randomness testing. The test results indicate that the method generates random output sequences, which correspond to produced keys being unique.

Table 5 gives the minimum P-Value of all possible states examined for random excursion and excursion variant testing. The predefined threshold value, i.e., P-Value ≥ 0.01 for all x states indicates that the output is random.

Table 4 Storage key space

Keys	1 K	10 K	100 K	1 M	10 M
Storage	63.5 KB	634.8 KB	6.2 MB	62 MB	619.9 MB

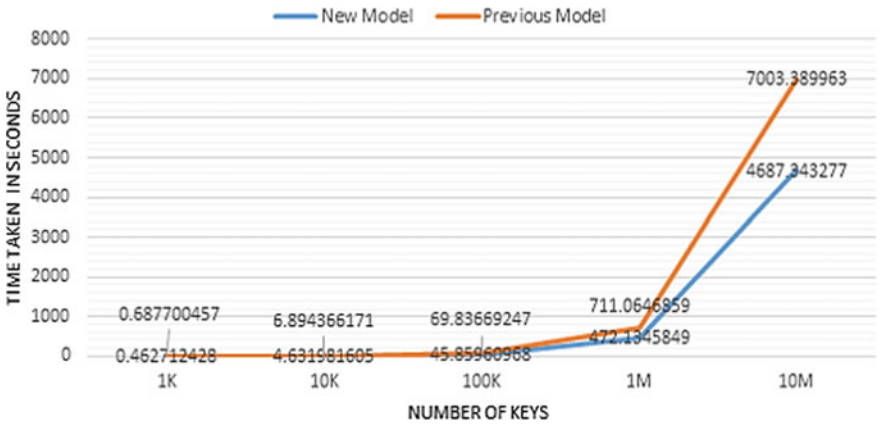

Fig. 13 Comparing performance analysis of previous and redesigned model

9 Conclusion

In conclusion, we describe a cyclic dynamic key generation technique capable of producing many unique keys. Most cryptanalytic assaults, such as dictionary attacks, ciphertext-only attacks, brute force, preimage, and replay attacks, are resistant to the method (and, by extension, the keys). The redesigned algorithm has better performance and full randomness compared to the previously designed model. Since this key generation technique is intended to be deployed in an IoT ecosystem, the prototype developed for the algorithm operates efficiently in resource-constrained environments.

Table 5 Randomness test result

S. No	Test type	Previous model result (P-Value)	Modified model with Blake2b (P-Value)
1	Frequency test	0.62202	0.5815
2	Frequency test within a block	0.0261	0.47955
3	Run test	0.6593	0.96586
4	Test for the longest run of ones in a block	0.6539	0.68468
5	Binary matrix rank test	0.0852	0.45961
6	Discrete Fourier transform test	0.1661	0.49789
7	Non-overlapping template matching test	0.0292	0.99986
8	Overlapping template matching test	NAN	0.06119
9	Maurer's universal statistical test	NAN	0.10592
10	Linear complexity test	0.03486	0.08011
11	Serial test	0.0523	0.49896
12	Approximate entropy test	1	1
13	Cumulative sums test (Forward)	NaN	0.5876
14	Cumulative sums test (Reverse)	NAN	0.79917
15	Random excursions test	0.0508	0.2109
16	Random excursions variant test	0.1573	0.58388

[a] NAN-Not a number. Comparing the NIST randomness testing results of previous and redesigned model.

References

1. Kimani K, Oduol V, Langat K (2019) Cyber security challenges for IoT-based smart grid networks. Int J Crit Infrastruct Prot 25:36–49
2. Samaila M, Neto M, Fernandes D, Freire M, Inácio P (2018) Challenges of securing Internet of Things devices: a survey. Secur Priv 1:e20
3. Abomhara M, Køien G (2014) Security and privacy in the Internet of Things: Current status and open issues. In: 2014 international conference on privacy and security in mobile systems (PRISMS). pp 1–8
4. Kumari S (2017) A research paper on cryptography encryption and compression techniques. Int J Eng Comput Sci 6:20915–20919
5. Mushtaq M, Jamel S, Disina A, Pindar Z, Shakir N, Deris M (2017) A survey on the cryptographic encryption algorithms. Int J Adv Comput Sci Appl 8:333–344
6. Ngo H, Wu X, Le P, Wilson C, Srinivasan B (2010) Dynamic key cryptography and applications. Int J Netw Secur 10:161–174

7. Singh K, Tomar S, Jain U, Hussain M (2020) Simplified and secure session key sharing for Internet of Things (IoT) networks. In: International conference on internet of things and connected technologies. pp 319–332
8. Pothumarti R, Jain K, Krishnan P (2021) A lightweight authentication scheme for 5G mobile communications: a dynamic key approach. J Ambient Intell Humanized Comput 1–19
9. Aumasson J, Neves S, Wilcox-O'Hearn Z, Winnerlein C (2013) BLAKE2: simpler, smaller, fast as MD5. In: International conference on applied cryptography and network security. pp 119–135
10. Dang T, Vo H (2019) Advanced AES algorithm using dynamic key in the internet of things system. In: 2019 IEEE 4th international conference on computer and communication systems (ICCCS). pp 682–686
11. Melki R, Noura H, Mansour M, Chehab A (2018) An efficient OFDM-based encryption scheme using a dynamic key approach. IEEE Internet Things J 6:361–378
12. Uchiteleva E, Hussein A, Shami A (2020) Lightweight dynamic group rekeying for low-power wireless networks in iiot. IEEE Internet Things J 7:4972–4986
13. Thirumalai C, Kar H (2017) Memory efficient multi key (MEMK) generation scheme for secure transportation of sensitive data over cloud and IoT devices. In: 2017 innovations in power and advanced computing technologies (i-PACT), pp 1–6
14. Moosavi S, Nigussie E, Virtanen S, Isoaho J (2017) Cryptographic key generation using ECG signal. In: 2017 14th IEEE annual consumer communications networking conference (CCNC). pp. 1024–1031
15. Ahmed S, Islam M, Nath T, Ferdosi B, Hasan A (2019) G-TBSA: a generalized lightweight security algorithm for IoT. In: 2019 4th international conference on electrical information and communication technology (EICT). pp 1–6
16. Akshay S, Vishnukumar B, Mohan V, Anand M (2018) Energy and performance analysis of raspberry pi with modern computing devices. Int J Eng Technol 7:777–779
17. Zhao C, Jegatheesan J, Loon S (2015) Exploring iot application using raspberry pi. Int J Comput Netw Appl 2:27–34
18. Rukhin A, Soto J, Nechvatal J, Smid M, Barker E (2001) A statistical test suite for random and pseudorandom number generators for cryptographic applications. (Booz-allen, 2001)
19. Bassham III L, Rukhin A, Soto J, Nechvatal J, Smid M, Barker E, Leigh S, Levenson M, Vangel M, Banks D et al (2010) Sp 800–22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications. (National Institute of Standards Technology, 2010)