SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V12I7P119

Volume 12 Issue 7, 238-245, July 2025

Original Article

An Efficient and Scalable Mutual Symmetric Key Establishment and Management for IoT Devices

Ahmed S. Alfakeeh

Faculty of Computing and Information Technology, King Abdulaziz University, Saudi Arabia.

Corresponding Author : asalfakeeh@kau.edu.sa

Received: 10 May 2025 Revised: 11 June 2025 Accepted: 12 July 2025 Published: 31 July 2025

Abstract *- Recently, the Internet of Things (IoT) has attracted the attention of many researchers due to its popularity in various applications. Connecting different capability devices in the Internet of Things architecture makes security a big challenge. Datagram Transport Layer Security Protocol is considered a standard for securing communication among Internet of Things devices by establishing a secret key. However, the default method, X.509, requires certificates and public key infrastructure that are more resource-consuming and not suitable for resource-constrained devices. On the other hand, datagram transport layer security supports the pre-shared key approach and raw pubic keys, which are lightweight but not scalable for such large networks. Hence, a scalable and lightweight mutual key establishment and management protocol is proposed for such a large number of resource-constrained IoT devices. The implementation of the proposed scheme in Contiki OS and on a real IoT platform shows its performance evaluation in terms of feasibility and scalability.*

Keywords *- Internet of Things, Authentication, Key establishment, Security, Secret key.*

1. Introduction

Transport Layer Security (TLS) protocol is one of the most famous and widely used security protocols in today's Internet [1, 2]. It is used in parallel with the Transmission Control Protocol (TCP) for reliable and secure communication. However, TLS cannot be used in the Internet of Things, as it usually uses the TCP approach, but many IoT applications use the User Datagram Protocol (UDP) for communication.

Keeping in view these limitations, IETF (Internet Engineering Task Force) developed Datagram Transport Layer Security (DTLS), which is a UDP-based on TLS and easily applicable in IoT applications [3, 4].

To establish a secure session using DTLS and TLS, client and server usually do a handshaking that is based on X.509 certificates and a corresponding verification infrastructure. However, this architecture is not suitable for a large number of resource-constrained IoT devices.

X. 509 certification is based on a central certification authority mechanism and also based on RSA (traditional PKI approach), which is not suitable for resource-constrained devices because of the high computational cost (multiplication and power operations). Also, a single point of failure is possible. Usually, sensor nodes use a preshared key approach to establish a secure session.

However, using the preshared key approach alone is not feasible and scalable for a large IoT network, especially in cases where two devices have no prior direct relationship with each other and belong to different servers. This is because the Internet of Things consists of billions of devices and requires a memory equal to the key size * billions of devices. So constrained devices do not have such a large memory [5, 6]. Also, disclosing the server's symmetric key to an unknown device is not a good approach, and using Public Key Infrastructure (PKI) is unsuitable. Hence, key establishment and management are important issues in IoTs. However, Kerberos-like protocols have been used for a long time to solve such issues that use Key Distribution Center (KDC) and Trusted Third Party (TTP) to establish a key between the two clients without revealing the secret shared keys of their servers to each other. In many Internet applications, TTP is used as an authorization server to grant access to the trusted clients to reach a server.

A similar Approach with some additional requirements is needed for IoT applications (for example, smart metering, building automation, personal health monitoring, and industrial control systems) where resource-constrained devices can get help from those authorization servers to generate a key and establish a session. But the size of messages and number of messages exchanged during the key establishment and session creation phase put an extra burden on resource-constrained devices and need to be optimized to

reduce this burden and improve the network performance and lifetime. Hence, introducing a Kerberos-like protocol for IoT applications is also not suitable.

This paper proposes an efficient and scalable mutual symmetric key establishment and management protocol between two IoT devices that have no direct relationship with each other. In this scenario, two IoT devices will contact their network managers for help in establishing a secret key between them.

This is because the key in the proposed scheme depends on both device parameters/share and network parameters/share to avoid any node replication attacks, sybil attacks, etc. First, the authentication phase is performed between the IoT device and the network manager, followed by key material generation. After successfully generating the key materials, an IoT device will be able to authenticate other IoT devices and establish a mutual symmetric key with them for secure communication.

The rest of this paper is organized as follows. The literature survey is presented in Section 2, while Section 3 presents the proposed algorithm for mutual key establishment and management protocol. Section 4 gives the complete details of the experimental testbed, while the results are discussed in Section 5, and finally, the paper is concluded in Section 6.

2. Related Work

Internet of Things (IoT) networks include a wide range of devices with limited processing, memory, and energy capabilities. Due to these constraints, traditional security protocols like TLS are not suitable for IoT environments. Instead, Datagram Transport Layer Security (DTLS) [5] is commonly used as it supports UDP-based communication. However, DTLS still requires a secure session setup through certificate-based handshaking, which is not practical for resource-constrained devices. To overcome these limitations, researchers have proposed several alternative approaches for authentication and key management in IoT.

Several works have enhanced authentication frameworks using cryptographic algorithms optimized for low-power environments. For example, Wang and Li [7] presented an improved IoT authentication protocol by integrating the Diameter framework with elliptic curve-based key agreement and lightweight encryption (AES/RC4). They also proposed a key management scheme based on multivariate quadratic polynomials, which improved scalability and security while reducing overhead. Similarly, Ju and Park [10] proposed a lightweight mutual authentication and key agreement protocol using elliptic curve cryptography and hash functions for cloud-based IoT, offering resistance against insider and impersonation attacks with low computational costs.

Another important aspect of IoT security is group key management. Abdmeziem et al. [8] proposed a blockchain-based group key management system that supports asynchronous IoT environments where device unavailability is common. Their protocol uses smart contracts and a reputation-based mechanism to ensure trust and secure consensus among distributed devices. Security features like Perfect Forward Secrecy (PFS) and Post-Compromise Security (PCS) were included with minimal overhead, making it suitable for real-world deployments.

To further resist physical attacks and support cross-domain communication, Mahmood et al. [9] introduced a blockchain and Physically Unclonable Function (PUF)-based key establishment protocol. Their method used an on-chain accumulator and cross-domain trust model, enabling devices from different domains to derive secure keys without revealing identities or requiring heavy computation. Similarly, Yang et al. [6] presented SAKMS, a key management protocol for 6TiSCH industrial wireless networks, using an improved elliptic curve algorithm. The scheme achieved faster key computation, implicit certificates, and dynamic key updates to minimize the risk of key leakage while maintaining compatibility with low-power devices.

Rana et al. [11] proposed an innovative key management system for cluster-based IoT networks using lightweight block cyphers. Their method used pre-distributed partial keys that are later combined into full encryption keys, reducing storage needs and allowing frequent key updates without transmitting the keys over the network. This approach provided scalability and reduced vulnerability to key exposure. While these approaches enhance authentication and key establishment in different IoT scenarios, most of them either rely on pre-shared keys, certificate-based mechanisms, or blockchain infrastructures, which may introduce added complexity, communication overhead, or dependency on external servers. In contrast, the approach proposed in this paper focuses on a scalable and lightweight mutual symmetric key establishment and management scheme without prior trust between devices. By engaging network managers and using both device-specific and network-specific parameters, the protocol improves security and resilience against attacks such as Sybil and node replication while ensuring minimal overhead and making it ideal for large-scale and heterogeneous IoT deployments.

3. Proposed Algorithm

Cryptography and key management are considered one of the main building blocks of security. However, it becomes more challenging if the network consists of resource-constrained devices (for example, sensors, smart objects in the Internet of Things). In the traditional security approach over the Internet, clients authenticate servers using their digital signatures while servers authenticate clients using their username and password. However, these approaches are not

suitable for resource-constrained devices used in IoT due to the lack of a keyboard and screen. Public Key Infrastructure (PKI) is computationally very expensive and unsuitable for use in constrained devices. Some efficient procedure to authenticate such constrained devices using lightweight certificates (e.g. based on ECC) is needed. However, the most favorable approach is to use a symmetric key Approach. Symmetric key establishment is done using pre-shared secret key materials or using a lightweight PKI approach based on Elliptic Curve Cryptography (ECC).

The main objective of the proposed approach is to develop a mutual authentication scheme that avoids impersonation attacks. For each new session establishment, a new encryption key should be used for security purposes, and this requires selecting a new additive factor that generates a fresh random key. Although a password-based method can simplify this process, it is vulnerable to keylogger attacks. In ad hoc networks, one-time authentication is usually sufficient; however, in IoT-based environments, authentication is required at the start of each new session, similar to how different accounts are accessed on the Internet using different devices.

Before explaining the proposed solution in detail, it is important to provide an overview of the network entities involved. The proposed network includes: (1) a client that utilizes the services of a server, (2) a server that provides services to clients, which may be either resource-constrained or resource-rich, (3) a Network Manager (NM) for each network, responsible for its management, (4) a Network Coordinator (NC) for each type of network (i.e., resource-constrained and resource-rich), and (5) a Trusted Third Party (TTP) that establishes trust between the NCs.

The architecture of the proposed network is illustrated in Figure 1. The authentication of IoT devices and establishing a secret symmetric communication key between them are the main security features of the proposed solution. The first authentication phase starts between the IoT devices, and once they authenticate each other successfully, they start the symmetric key establishment phase.

3.1. Key Pre-Distribution Phase

Before the network deployment, each device is assigned some authentication and key generation materials offline. More specifically, each node is assigned (1) a special one-time authentication code, (2) an elliptic curve point generator, (3) an authentication material generation function and (4) a symmetric key generation function.

3.2. Authentication

Before going into the details of the authentication procedure performed by the IoT devices to authenticate each other, the authentication materials are first generated by the device and its network manager.

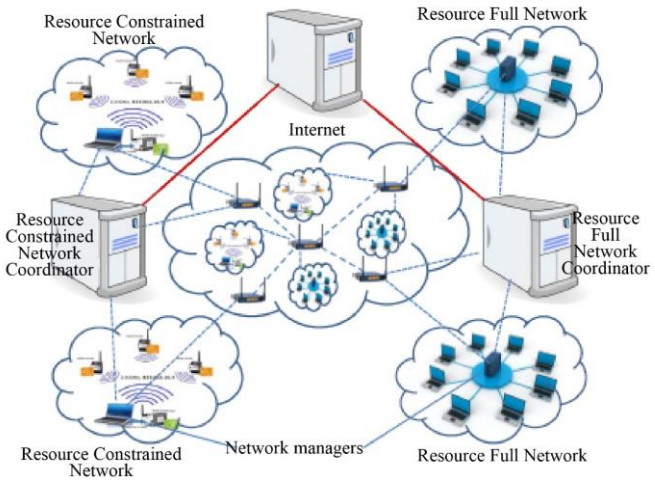

Fig. 1 Proposed network architecture

3.2.1. Authentication Material Generation

Once the network is deployed, the network manager starts broadcasting Hello messages to learn about the network devices nearby. Each Hello message also consists of a special authentication code assigned to the NM before the network deployment to authenticate itself to its network devices. After receiving Hello messages from network devices, each device sends a joining message including its own special authentication code to authenticate itself to the NM. Upon receiving a Hello response/joining message from network devices, NM selects an elliptic curve EP(a,b) and a point eNM and sends this information to the network device. After receiving EP(a,b) and eNM, each device randomly chooses an additive factor dD and a point eD over EP(a,b) and calculates W as

$$W = dDeNM \quad (1)$$

Each device sends W and eD back to its NM, where the NM calculates V after selecting an additive factor dNM as

$$V = dNMeD \quad (2)$$

NM sends V to each particular network device from which it received eD. This W and V help the NM and a network device to calculate mutual authentication materials generation (i.e. E1 and E-2). The NM calculates E1 as

$$E1 = eNM + V \quad (3)$$

and the network device calculates E2 and shares it with NM as

$$E2 = dDV + W \quad (4)$$

Now, NM has EP(a,b), E1 and E2 for each device and registers this information with its NC. Figure 2 represents all the necessary steps involved in generating authentication-related materials.

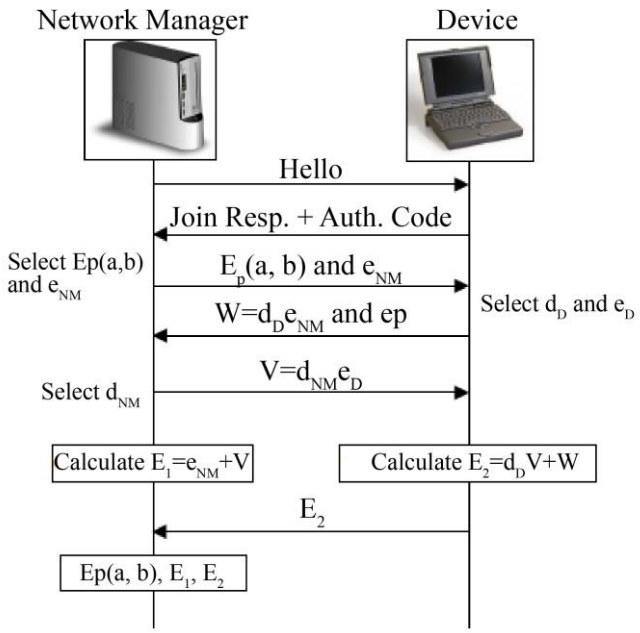

Fig. 2 Authentication material generation by a device and its network manager

3.2.2. Authentication Phase

After successfully generating authentication materials, the two devices are able to authenticate each other. The authentication procedure between any two IoT entities is the same, i.e., the same authentication procedure is followed by the network managers to authenticate each other, between the devices belonging to different networks, and between the network manager and its network coordinator. To better understand the authentication procedure, two devices that belong to two different networks are considered. Suppose device-1 wants to establish a secure session with device-2, it sends a session establishment request to its NM1. Suppose NM1 does not have the security credentials of NM2 of device-2. In that case, NM1 contacts the network coordinator of NM2 to get its security credentials after receiving the NM2 credentials (i.e. EP(a,b), E1 and E2), NM1 becomes able to authenticate NM2 and sends device-1 credentials to NM2 securely and requests device-2 credentials. Once NM2 receives a message from NM1, it first authenticates NM1. If NM2 does not have the NM1 security credentials, it contacts the network coordinator of NM1 to get its security credentials. After successful authentication, NM2 send the device-2 credentials to NM1. NM1 sends the received credentials of device-2 to device-1, while NM2 sends the received credentials of device-1 to device-2. Once a device receives the security credentials of another device (i.e. EP(a,b), E1 and E2), it generates C1 as

$$C1 = rE1 \quad (5)$$

and C2 as

$$C2 = Nonce + rE2 \quad (6)$$

After successfully calculating C1 and C2, each device sends them to the other communicating device. After receiving C1 and C2, each device extracts the sent Nonce from it.

$$Nonce = C2 - dNC1 \quad (7)$$

If Nonce is calculated correctly, it is sent back to the sending device to prove its authenticity (i.e., the sending device used the correct credentials of the receiving device and the receiving device has correctly recovered the Nonce from C1 and C2). Figure 3 represents the necessary steps in the authentication process of two devices belonging to two different networks.

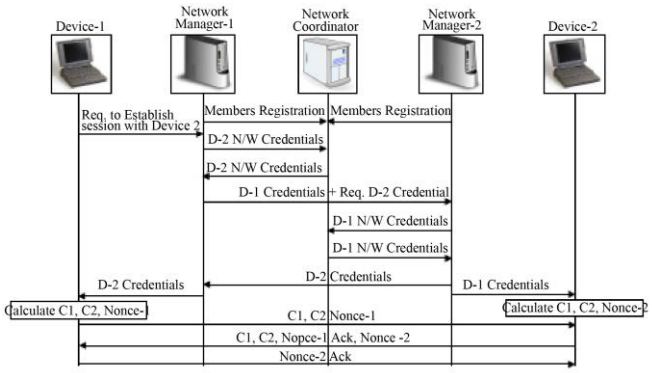

Fig. 3 Authentication process of devices

3.3. Symmetric Key Establishment

After successfully authenticating each other, two devices establish a mutual symmetric key for secure communication, a function of the device and network share.

$$Key = f(Device/Share + Netowrk/Share) \quad (8)$$

In the proposed approach, this symmetric key consists of four parts, namely (1) device-1 share, (2) device-1's network manager share, (3) device-2 share and (4) device-2's network manager share. The dependence of symmetric key on the mentioned network entities shares makes it difficult for an attacker to launch any impersonation attacks or guess those shares. To establish a symmetric key, device-1 sends a key establishment request to device-2 along with its key share aD1V1, where aD1 is a randomly selected constant by device-1. Similarly, device-2 also sends its own key share, aD2V2, during the key establishment response.

Once the devices receive the device shares of the key from each other, they send a request to the network managers to get the network share. Device-1 sends a request to NM2 while Device-2 sends a request to NM1. Now the NM2 sends the network manager share of key bNM2W2 in its response to device-1, while NM1 sends its own network manager share of key bNM1W1 to device-2. After receiving all the key shares from each device, they establish a mutual symmetric key as

$$Key = aD1V1 + aD2V2 + bNM1W1 + bNM2W2 \quad (9)$$

Figure 4 represents the necessary steps in the mutual symmetric key establishment process between two devices belonging to two networks.

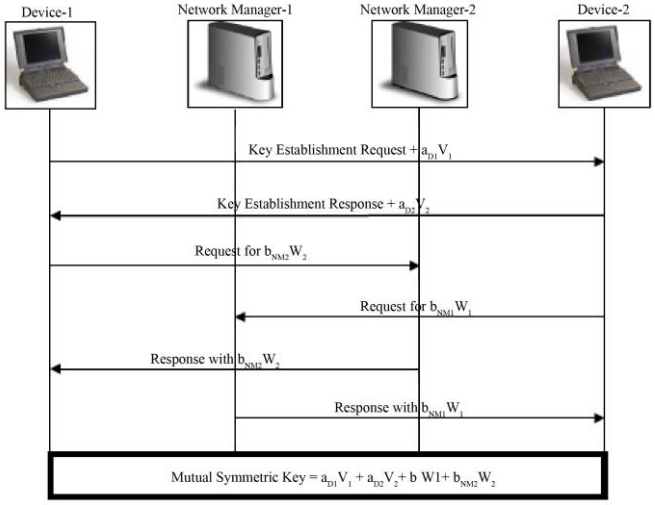

Fig. 4 Symmetric key establishment between two IoT devices

3.4. Additional Key Management Services

3.4.1. Clock Synchronization Issue

It is not a good practice to make the decision about freshness or expiration of a key based on date and time. This is because most resource-constrained devices do not have a reliable means of measuring real time and suffer from clock synchronization especially when they sleep for a long period to save their energy and increase the network lifetime. However, the platform used (i.e. CC2538DK of Texas Instruments) has a real-time clock synchronization mechanism, and freshness or expiration decision of keys can be done on date and time. However, not all IoT devices have such a mechanism; hence, this paper does not focus on this approach to key freshness or expiration.

3.4.2. Key Freshness

There are a number of attacks where a compromised key can be used at later stages, such as replay attacks. Hence, the freshness of the key is an important factor that needs to be considered and addressed in key establishment and management algorithms. To this aim, the proposed algorithm uses network share and device share, where random numbers are multiplied (meaning addition of a point multiple times based on that random number) with the point to generate a secret symmetric session key. Once the session expires, IoT devices want to establish a new session; they generate a new symmetric session key.

3.4.3. Key Expiration

Each security key needs to expire after some time to secure the system and prevent replay attacks. In the proposed scheme, the security key based on time stamps (like date and time approach) does not expire, but the key expiration in the proposed scheme is based on the session, irrespective of its duration. For example, if a device establishes different sessions at different time intervals with other devices, it will use different keys. Since these sessions may have different durations, each key will have a different expiration time.

3.4.4. Key Revocation

Although the key expiration depends on the session duration in the proposed scheme, there is still a possibility of key compromise during the session. This is because if one of the two communicating IoT devices gets compromised, it violates the mutual agreement. In this case, key revocation needs to be done early, and other communicating IoT devices must send a key revocation message to both the network managers from whom they got network shares during the key establishment phase.

4. Implementation

To evaluate the proposed key establishment and management scheme, two resource-constrained IoT devices connected directly with their network managers are implemented, and the network managers are connected directly to the network coordinator. All the devices were in the communication range of each other for experimental tests to keep the network architecture simple. Figure 5 shows the experimental setup of the proposed network architecture.

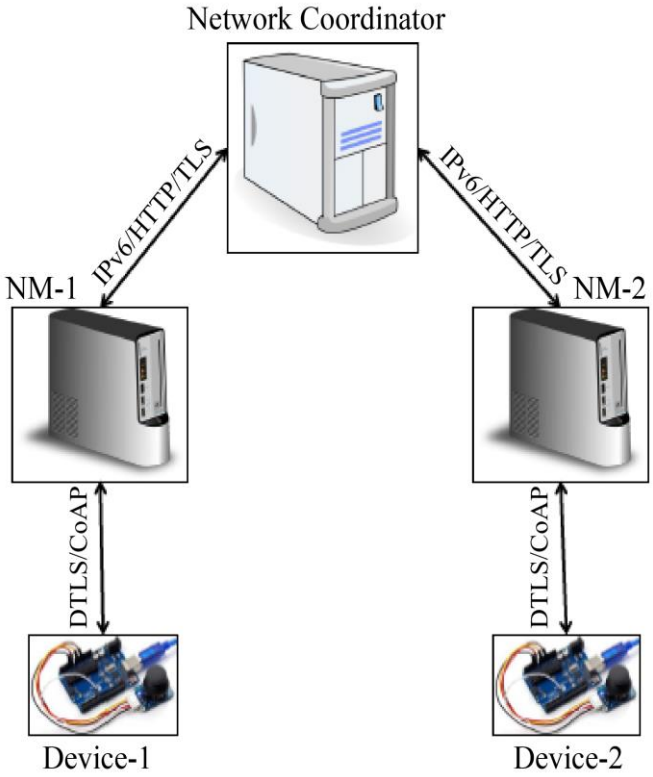

Fig. 5 Experimental setup

The implementation details are as follows: (1) The network coordinator is implemented as a Java web server application, and (2) the network manager is implemented as a Java application on a laptop. (3) IoT devices are implemented in C on constrained devices.

The network coordinator, which is implemented as a web service application, accepts the members' registration and network credential requests from each network manager as an HTTP request over a secure TLS connection. The network manager is connected to the resource-constrained devices through a serial port using the Serial Line Internet Protocol (SLIP) protocol. At the same time, the resource devices are connected to each other through IEEE 802.15.4 radio links.

4.1. Network Coordinator Implementation

A Network Coordinator that is implemented as a web server provides service to the network managers. It registers the member resource-constrained devices of each network manager to keep track of those devices and avoid any node replication attacks. It also provides some security credentials (explained in III) to each network manager that help in the authentication and key establishment phases with the network coordinator and the resource-constrained devices.

However, from the implementation point of view, the network coordinator accepts each message as a JSON object that includes the details of the network managers. The response is also a JSON object that includes the confirmation or any correction to the member registration request or the network credentials to the network credentials request. TLS approach is used at this level based on X.509 protocol for implementation purposes because the main objective is to evaluate the proposed algorithm (only key establishment part) on resource-constrained devices.

4.2. Network Manager Implementation

The network manager is also implemented as a Java application. It performs the following functions: (1) registers the member resource-constrained devices with the network coordinator and gets security credentials from the network coordinator, (2) establishes a secure link with resource-constrained devices and helps them to authenticate each other and establish a mutual symmetric key. In the proposed implementation, the network manager performs a DTLS handshaking with resource-constrained devices through the provided key materials and sends a CoAP request to the device (like an HTTP request is sent by a device to a server when a URL is entered).

4.3. Resource Constrained Device

The main objective of implementation is to check the real performance of the proposed algorithm over constrained devices. So the algorithm is implemented on the CC2538DK of Texas Instruments, which has enough memory and computational power to perform all the tasks. It also supports the IEEE 802.15.4 radio link through which the devices communicate directly. CC2538 also has cryptographic accelerators and a random number generator that plays an important role in the proposed key generation algorithm.

For practical implementation, the CC2538DK development kit of Texas Instruments is used, which contains a CC2538F512RKU processor, RAM of 32 KB, ROM of 512 KB, operate at a 32MHz clock frequency and at a voltage of 2.1 V. The current rating is 13 mA and has a 32.768 kHz oscillator frequency. It also has a light sensor, 4 LEDs and an accelerometer.

1) Contiki OS: Contiki Operating System (OS) is specifically designed for IoT applications and is an open-source OS. Contiki OS supports Internet Protocols, including IPv6, TCP, and UDP. It also has a built-in CoAP protocol and supports server/client architecture with minimal code size.

2) DTLS Library: A tiny DTLS library is used to implement DTLS functionality. This is implemented in the C language and is suitable for resource-constrained devices. However, it has been extended to support the proposed algorithm. For example, a hardware accelerator is used to generate random numbers instead of a software-based procedure. Also, the STLS session handler is modified so that it searches for the existing session, and the DTLS event close is added.

3) CoAP and DTLS Integration: CoAP is integrated with DTLS to secure CoAP functionality. However, this security is flexible and can be enabled and disabled using an added SECURE flag. A callback method has also been implemented to handle the CoAP secure messages. Whenever a message is received by a CoAP receiver, it checks whether to process the message securely or not. If it is to be sent without security, the data message is handled as a normal CoAP message.

5. Implementation

To show the performance of the proposed algorithm, the network setup shown in Figure 5 is evaluated to check the memory, time and energy overhead.

1) Memory Cost: To check the code size and memory utilization, arm-none-eabi-size is used. This is because arm-none-eabi-size gives the information about both RAM and ROM utilization, and is also included in the ARM processor utility toolchain.

The total memory overhead of the proposed algorithm is 1537 bytes. Among these 1537 bytes, 24 bytes are reserved for global variables, while static information consumes 48 bytes, and the remaining 1465 bytes are consumed by code.

The total memory occupied by the tiny DTLS library is 21592 bytes, including all the necessary changes in it. Detailed investigations showed that 19368 bytes are occupied by program code, while constant data occupied 140 bytes, and 2084 bytes are occupied by global variables.

2) Time and Energy Cost: To calculate the total time taken by computation and energy consumed, the Contiki Energest module [13] is used. This is because it is based on a time clock and also measures the time of different components separately. For example, CPU time, transmission time, etc. The total energy consumed is calculated as

$$\text{Energy} = V * I * t \quad (10)$$

Where V is the voltage of the power supply, I is the average current consumed, and t is the time during which the modules are active. Its value is taken from the Energest module of Contiki. In this way, the energy consumed by each module is calculated and summed up to observe the total energy consumption of the processor and radio transceiver. Also, the total energy consumed during the authentication, key generation and establishment phases is calculated separately and shown in Table 1.

Table 1. Energy consumption during authentication and key generation phase

Function	Energy Consumed (µJ)
Authentication	32
Key Generation	23

The energy consumption of the proposed scheme is also compared with the existing approach [14], which uses the same platform for evaluation purposes, but the proposed solution performs better than [14]. Figure 6 shows the comparison of the energy consumption of the proposed approach with [14].

3) Security Analysis: Although cryptography is considered one of the main fundamental pillars of any security system, the management and authenticity of those cryptographic keys are more challenging in an IoT environment. Hence, the exchange of cryptographic key material must be secure and verifiable.

In the proposed algorithm, a cryptographic key not only depends on the two communicating devices but also depends on their network managers and their shares. This makes the proposed algorithm more robust against attacks.

For example, an attacker would not be able to establish a secret key with any other IoT device because the attacker would not be able to authenticate itself to the network manager and hence would not be able to get the required key share from the network managers.

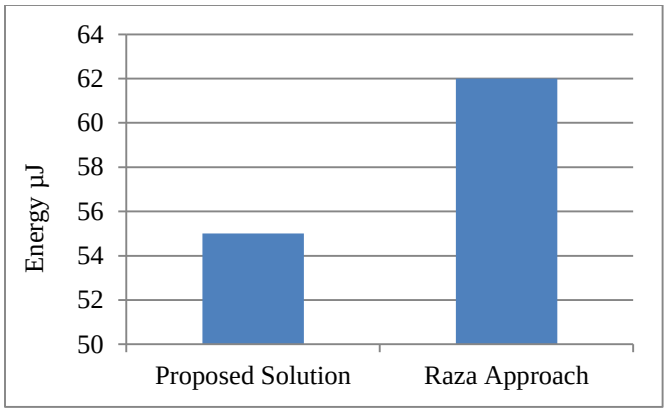

Fig. 6 Overall energy consumption during the key generation process

The AVISPA tool (Automated Validation of Internet Security Protocols and Applications) was used to evaluate the security strength of the proposed key management approach for IoT devices. AVISPA is a widely used platform that helps automatically test the security of network protocols. It offers a flexible way to define protocols and their security goals, and includes multiple analysis engines that apply advanced techniques to find possible vulnerabilities.

The key management scheme was implemented using AVISPA and tested against common types of attacks supported by the tool, including OFMC (On-the-Fly Model-Checker) and CL-AtSe (Constraint-Logic-based Attack Searcher). OFMC dynamically builds a model of the protocol's behavior during the analysis process, using symbolic methods to manage complex states. CL-AtSe, on the other hand, translates the protocol into a set of logical constraints and searches for possible security flaws automatically. Both tools carry out the analysis without needing manual intervention. Results are shown in Table 2.

Table 2. Avispa simulation results

Technique	Summary
OFMC	SAFE
CL-AtSe	SAFE

6. Conclusion

Although the DTLS approach is becoming a security standard for IoT devices, it requires a PKI certification approach that is very resource-consuming in terms of computation and is considered not suitable for resource-constrained devices that are also part of an IoT network. A scalable and mutual authentication and key establishment protocol that does not require any certification authority and is computationally lightweight is proposed in this work. The implementation of the proposed protocol over resource-constrained devices in an IoT network showed its feasibility and adaptability in such a highly dynamic, heterogeneous network. Future work aims to develop and implement similar approaches to other cybersecurity protocols, such as IPsec/IKE.

References

[1] Matheus K. Ferst et al., "Implementation and Analysis of a Secure Communication with SunSpec Modbus and Transport Layer Security Protocols for Short-Term Energy Management Systems," *IEEE Access*, vol. 13, pp. 105183-105198, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[2] Mingping Qi, and Chi Chen, "HPQKE: Hybrid Post-Quantum Key Exchange Protocol for SSH Transport Layer from CSIDH," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 2122-2131, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[3] Geovane Fedrecheski, Mališa Vučinić, and Thomas Watteyne, "Performance Comparison of EDHOC and DTLS 1.3 in Internet-of-Things Environments," *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, Dubai, United Arab Emirates, pp. 1-6, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[4] Min Shi et al., "A Formal Analysis of 5G EAP-TLS Protocol," *IEEE Transactions on Networking*, pp. 1-13, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[5] Muhammad Rana, Quazi Mamun, and Rafiqul Islam, "Enhancing IoT Security: An Innovative Key Management System for Lightweight Block Ciphers," *Sensors*, vol. 23, no. 18, pp. 1-25, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[6] Wei Yang et al., "SAKMS: A Secure Authentication and Key Management Scheme for IETF 6TiSCH Industrial Wireless Networks Based on Improved Elliptic-Curve Cryptography," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 3, pp. 3174-3188, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[7] Qing Wang, and Haoran Li, "Application of IoT Authentication Key Management Algorithm to Personnel Information Management," *Computational Intelligence and Neuroscience*, vol. 2022, pp. 1-11, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[8] Mohammed Riyadh Abdmeziem, Amina Ahmed Nacer, and Nawfel Moundji Deroues, "Group Key Management in the Internet of Things: Handling Asynchronicity," *Future Generation Computer Systems*, vol. 152, pp. 273-287, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[9] Khalid Mahmood et al., "Blockchain and PUF-Based Secure Key Establishment Protocol for Cross-domain Digital Twins in Industrial Internet of Things Architecture," *Journal of Advanced Research*, vol. 62, pp. 155-163, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[10] Sieun Ju, and Yohan Park, "Provably Secure Lightweight Mutual Authentication and Key Agreement Scheme for Cloud-Based IoT Environments," *Sensors*, vol. 23, no. 24, pp. 1-25, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[11] Muhammad Rana, Quazi Mamun, and Rafiqul Islam, "Enhancing IoT Security: An Innovative Key Management System for Lightweight Block Ciphers," *Sensors*, vol. 23, no. 18, pp. 1-25, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[12] "*CC2538EM Reference Design*," Technical Report, 2012. [Publisher Link]

[13] Adam Dunkels et al., "Software-Based On-Line Energy Estimation for Sensor Nodes," *Proceedings of the 4th Workshop on Embedded Networked Sensors*, Cork Ireland, pp. 28-32. [CrossRef] [Google Scholar] [Publisher Link]

[14] Shahid Raza et al., "S3K: Scalable Security with Symmetric Keys—DTLS Key Establishment for the Internet of Things," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 3, pp. 1270-1280, 2016. [CrossRef] [Google Scholar] [Publisher Link]