JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 13(2), JUNE, 2025
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.com.ng

Secure Energy-Efficient Device-to-Device Communication for IoT in Smart Cities

[1]Joel Haruna, [2]Muhammad Bashir Abdulrazaq, [3]Basira Yahaya
[1]*Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria*
[2&3]*Computer Engineering, Ahmadu Bello University, Zaria – Nigeria*

ABSTRACT
This study introduces a novel compressed secure Device-to-Device Communication scheme (CSDDCS) based on Elliptic Curve Cryptography (ECC) to enhance energy-efficient communication in IoT system, particularly in smart cities. The integration of IoT and cloud computing in smart cities necessitate resource Optimization, and enabling direct device-to-device communication is crucial. The CSDDCS employs continuous authentication using compression ECC, where the registration request are encoded from the device during it registration with the server. The authentication message is decoded and analyzed by the server to verify the device's key component before forwarding the authentication to the next device. Performance evaluation revealed that the CSDDCS outperforms existing schemes with significant reduction in storage requirement, improved communication cost, reduce transmission time, albeit with a slight 2.28% decrease in computational time. The CSDDCS present an efficient and secure solution for IoT device-to-device communication, making it suitable for practical development in smart cities and similar environments.

ARTICLE INFO
Article History
Received: January, 2025
Received in revised form: February, 2025
Accepted: May, 2025
Published online: June, 2025

KEYWORDS
Device-to-Device Communication, Elliptic Curve Cryptography, Energy efficiency, point compression, IoT

INTRODUCTION

The Internet of Things (IoT) is a groundbreaking technology enabling global connectivity between people and a myriad of smart devices. It was introduced by Kevin Ashton in 1999 and has evolved single-function gadgets into smart devices like Apple Watch and smartphones [1]. Smart devices in the IoT possess internet connectivity, sensors, and actuators to sense and respond to environmental changes. Data collected from these devices is sent to central servers for processing, underlining the immense potential of IoT. IoT is widely used in smart homes, industrial applications, smart cities, healthcare, vehicular networks, and more. Security is a growing concern as malicious actors exploit IoT devices for internet attacks [2].

Efficient authentication of IoT devices is crucial, and Elliptic Curve Cryptography (ECC) is emerging as a promising solution due to its advantages in Device-to-Device (D2D) communication. ECC, a lightweight cryptographic method, provides strong security with shorter key sizes compared to other algorithms like RSA and DSA. ECC's efficiency makes it a compelling choice for securing IoT networks [2, 3]. Reducing transmission overhead is critical to minimize energy consumption, particularly in smart city environments. The compression technique optimizes D2D communication efficiency, promoting streamlined data exchange among devices without compromising security.

The study of [2] introduced an efficient mutual authentication protocol for smart cities. It comprised four essential modules: system setup, key generation, initialization, and authentication. Their system showed impressive speed but lacked device anonymity, making it vulnerable to server impersonation attacks.

In [3], a secure authentication scheme for IoT devices using secret parameters delivered over a private channel is proposed. Devices

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

securely stored identity and secret key values, selecting a cryptographic function and sending computed parameters for server authentication. Performance evaluation demonstrated improved device authentication with efficient cryptographic operations. However, communication costs relative to device storage capacity were not considered in the evaluation.

In the article [4], a secure mutual authentication mechanism for IoT, utilizing an ID verifier based on elliptic curve cryptography was introduced. The scheme involved a setup phase where parameters and keys were generated, and an authentication phase for mutual authentication. The performance evaluation demonstrated the scheme's resilience against various attacks and efficient message exchange. However, improving communication efficiency by implementing compression techniques was suggested to further enhance the method.

The study of [1] proposed an energy-efficient elliptic curve cryptographic-based extension for device-to-device communication in IoT local networks, utilizing a server-controlled hybrid model. The process involved three phases: registration, authentication between servers, and device authentication. Energy consumption was relatively high for the initiating device due to multiple operations. The method showed resilience to attacks but could benefit from compression techniques to reduce communication and storage costs, improving overall energy efficiency in device-to-device communication.

An authentication scheme for distributed IoT, ensuring secure communication and scalability was introduced in [5]. It comprised two modules: one using JWT-based authorization with symmetric cryptography and a response game challenge, and the other employing ABE with access control and key agreement for encryption. The method showed significant improvements, particularly in device authentication, with reasonable storage requirements. However, it had security vulnerabilities related to user and device anonymity and did not address communication overhead effectively.

The study of [6] proposed a secure and anonymous IoT authentication scheme using ECC-based pairing. The process involved registration and authentication phases for devices joining the network. The method improved security and demonstrated better performance than other methods for cloud authentication. However, it was not suitable for device-to-device (D2D) authentication due to high computational and storage demands, similar to RSA encryption.

In [7], a secure and efficient lightweight authentication method for IoT was designed, ensuring secure communication between devices and the server. Implementation on embedded devices using Pairing-Based Cryptography (PBC) library demonstrated its lightweight nature, reducing computation time by 0.933 ms, communication cost by 1408 bits, and storage cost by 384 bits. However, the protocol was limited to communication between devices and the server.

The article [8] proposed a secure and efficient authentication scheme for IoT in smart manufacturing, focusing on communication between devices and the server. It employed mutual authentication and key agreement using cryptographic hash functions, ECC, bitwise XOR, and PUF to counter physical tampering attacks and resource constraints. The scheme was formally proven secure and showed resistance to attacks but had higher communication overhead (3616 bits for five messages) compared to some other schemes.

The literatures reviewed underscores the persistent challenge of computational overhead in the context of IoT, primarily due to the constraints of power and computing capabilities in IoT devices. The central issue revolves around the imperative to streamline resource consumption and expedite response times for authentication requests within IoT environments, especially when dealing with resource-constrained devices. Current methods tend to exhibit elevated power consumption during data transmission, which contributes to heightened energy usage and extended response times for device authentication.

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 13(2), JUNE, 2025
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.com.ng

This study specifically focuses on energy-efficient ECC with a two-point compression technique to optimize resource consumption and improve device-to-device communications in IoT environments, particularly in smart cities. The study's objectives include applying ECC for secure device-to-device communication, developing point compressions to create a compressed secure device-to-device communication scheme (CSDDCS), and evaluating its performance based on communication cost, storage cost, and execution time.

This paper comprises five sections. Section 2 provides a detailed overview of the D2D communications in IoT environments and how the devices were derived. Section 3 outlines the development of the Compressed Secure Device-to-Device Communication Scheme (CSDDCS). Section 4 presents the outcomes and engages in a discourse about the simulation results. Finally, Section 5 wraps up the study by summarizing the contributions of

IoT Device to Device Communication

Within the IoT, two categories of Device-to-Device (D2D) communication exist: standalone D2D and network-assisted D2D, as noted in Dang, Pham [1]. These designs differ in their use of an assisting infrastructure that manages resource consumption and communication. In the case of Network-Assisted D2D, a gateway is vital for device operations, and cellular networks often serve as the connection medium, which can be costly in certain application areas. On the other hand, standalone D2D operates with devices employing mechanisms like Bluetooth for initiating communication with nearby, short-range devices Hammad, Badshah [8]. Figure 1 depicts a Smart City scenario in which the system monitors and collects environmental data for urban planning and public service, such as air quality, noise level, and traffic congestion [2].

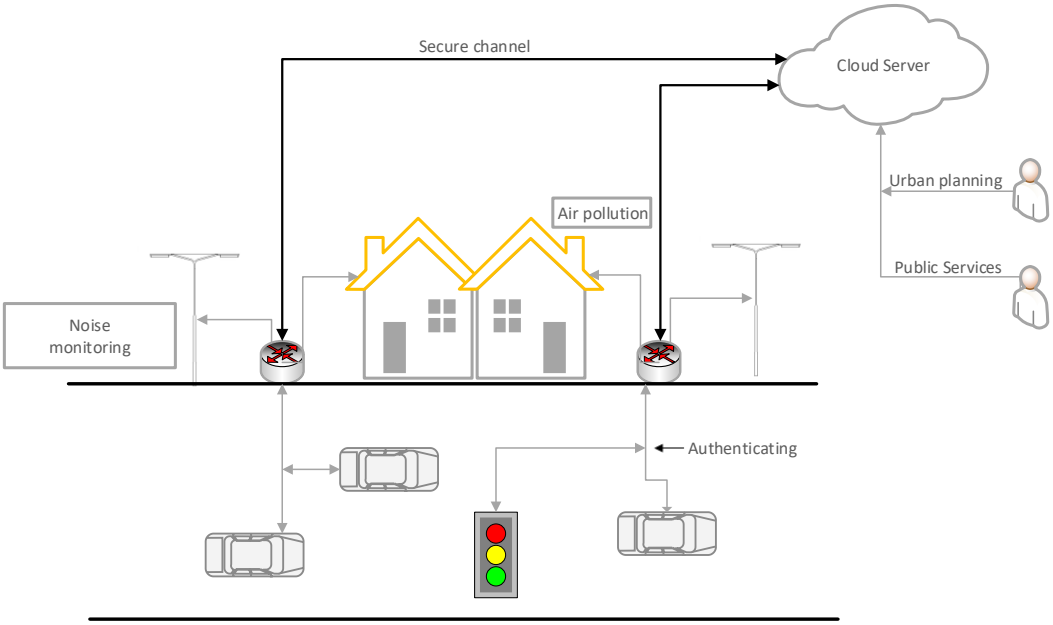

Figure 1. IoT Smart City Application

In the Smart City context, there are four integral components: "things" encompassing sensors and tags, intermediate nodes, a cloud server, and users. The "things" are resource-limited devices responsible for collecting

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

environmental data and transmitting it to the cloud server. To ensure data trustworthiness, these "things" must undergo authentication. Intermediate nodes, such as routers and gateways, act as intermediary devices connecting the "things" to the cloud server. The cloud server receives data from these "things," processes the data, and offers services to both authorities and the public. In this context, authentication is crucial to safeguard sensitive information from unauthorized access [6]. Users, who utilize the services provided by the cloud server, play a vital role in understanding the environment and potentially making informed decisions.

Authentication

Authentication is a process designed to confirm the true identity of an entity, with the primary goal of gaining access to protected resources like systems, applications, files, and networks. This process typically commences when a user reaches a login portal and provides their credentials, such as a username and password, or other secure tokens. The provided credentials are compared to the data stored in an associated directory services database, serving as the authoritative source for verifying user identities. Access is granted when the provided data aligns with the stored credentials, while denial occurs when they don't match. This process comprises two steps: identification, which asks who the user is, and authentication, which validates their true identity [1, 3, 9].

Authentication relies on four fundamental factors:

a) "What you know": This aspect pertains to cognitive data, such as passwords, Personal Identification Numbers (PINs), and security codes, which should remain confidential and known only to authorized users.
b) "What you have": This factor focuses on items that are exclusively accessible to authorized users, including tokens, smart cards, radio frequency identification (RFID), and photo identification.
c) "What you are": This factor requires unique human characteristics for authenticating authorized users, including face recognition, fingerprint scanning, palm scanning, retina scanning, iris scanning, and voice recognition.
d) "Where you are": Authorized users are identified based on location information, such as internet protocol addresses (IP-addresses), cell towers, and Global Positioning System (GPS).

Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) represents a state-of-the-art public-key encryption technique that leverages mathematical elliptic curves, known for generating smaller, faster, and more efficient cryptographic keys. Notably, ECC is employed in systems like Bitcoin due to its lightweight nature [3,17-19]. What sets ECC apart is its ability to achieve the same level of security as other public cryptosystems while using shorter encryption keys. This feature not only conserves significant energy when implemented on hardware but also makes ECC resilient to cryptographic attacks based on factorization and discrete logarithm, which are ineffective against ECC [10, 11]. In the context of ECC, an elliptic curve is a planar curve defined over a finite field, consisting of points that must meet specific criteria as detailed in the following equation [3, 9].

$$y^2 = x^3 + ax + b \quad (1)$$

Where, a, b are either rational numbers or integers and is a module computation done with some integer n extended by a “point at infinity”, usually indicated as ∞ (or 0) that can be regarded as being, at the same time, at the very top and very bottom of the y-axis.

Any point on the curve can be mirrored over the x-axis in this elliptic curve cryptography example, and the curve will remain the same. In three spots, any non-vertical line will intersect the curve [1]. The trap door function of ECC is like a mathematical game pool that start with an arbitrary point on the curve using a dot function in

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 13(2), JUNE, 2025
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.com.ng

order to get the new point. The process uses the dot function as illustrated in Figure 2 repeat continuously around the curve until the last point [12].

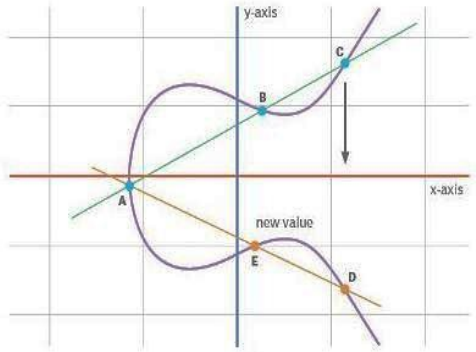

Figure 2. Elliptic cryptographic curve

In Figure 2, ECC is grounded in the properties of a set of values where mathematical operations can be performed on any pair to produce a third number. This process generates points on a graph, depicted as a green line and three blue dots in the diagram, denoted as A, B, and C. Each point on the curve can be obtained by multiplying a point by a number, resulting in point C. Then, point D is derived by reflecting point C across the x-axis. Subsequently, a line connects point D back to point A, intersecting at point E. This sequence can be repeated a specified number of times, defined as 'n', within a set maximum value. 'n' represents the private key, dictating how many iterations of the equation occur and concluding with the final value used for data encryption and decryption. The maximum value defined in the equation is linked to the chosen key size.

Point Compression

Point compression is a technique that reduces the data required to represent points on an elliptic curve, resulting in decreased data transfer and storage needs on smart devices. Instead of transmitting an ordered pair to convey a relevant cryptographic elliptic curve point, it's possible to send just the x-coordinate and a single bit of information. With the x-coordinate and this bit, the corresponding y-coordinate can be calculated [13, 14]. This compression method finds utility in multi-user broadcast authentication in wireless sensor networks and message authentication in vehicular ad-hoc networks. ECC-based pseudorandom generators, which have two elliptic curve points in their internal state, can benefit from compression when they are infrequently used, allowing for decompression as needed. Given an elliptic curve:

$$y^2 = x^3 + ax + b = f(x) \quad (2)$$

$$p_1 = (x_1, y_1) \quad (3)$$

$$p_2 = (x_2, y_2) \quad (4)$$

The points in equations **Error! Reference source not found.** and **Error! Reference source not found.** can be represented in general by:

$$(x_1, x_2, \alpha = y_1 + y_2) \quad (3)$$

Then,

$$y_1 = \frac{\alpha}{2} + \frac{x_1 - x_2}{8\alpha}(3(x_1 + x_2)^2 + (x_1 + x_2)^2 + 4\alpha) \quad (4)$$

$$y_2 = \alpha - y_1 \quad (5)$$

This is easily seen by remarking that:

$$y_1 - y_2 = \frac{y_1^2 - y_2^2}{y_1 + y_2} = \frac{f(x_1) - f(x_2)}{\alpha} \quad (6)$$

And knowledge of α equals y_1 + y_2 will readily yield y_1, y_2 by linear algebra. This algorithm works, as long as α ≠ 0.

Elliptic Curve Cryptography for Secure Device-To-Device Communication

Elliptic curve cryptography (ECC) is a public key cryptographic algorithm that is well-suited for secure device-to-device (D2D) communication. It is presumable that there is a reliable server and that several devices $D_i (D_i \in D)$ desire to connect to it. The server alone is in possession of the secret value X. Other public parameters include an elliptic curve E, two cryptographic hash functions H and h, and a generator G on E. Unlike h, which maps any string to a string l_h-bit, H maps any string to a string l_H-bit. In addition to being a generator of this group, G is an additive group on E [3]. The procedure for authentication is shown in Figure 3. The protocol in Figure 3 was designed and tested

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 13(2), JUNE, 2025
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.com.ng

to ensure its suitability for the IoT environment, while also demonstrating resilience against various potential attacks. To facilitate a better understanding of the protocol, Table 1 provides a comprehensive description of the notations used throughout the creation and implementation of the scheme.

Two phases of registration and authentication are shown in Figure 3. When an embedded device has to register with the server during the registration phase, it selects a special ID_4 and sends ID_1 to the server. The server creates a random number R_1 of length l_H-bits after receiving the request from D_4 and then computes CK_1, CK_i', T_1, A_i and A_i' as follows [1, 3]:

$$CK_i = h(R_i \parallel X \parallel \text{EXP\_Time} \parallel ID_i) \quad (7)$$
$$CK_i' = CK_i \times G \quad (8)$$
$$T_i = R_i \oplus H(X) \quad (9)$$
$$A_i = h(R_i \oplus H(X) \parallel CK_i') \quad (10)$$
$$A_i' = A_i \times G \quad (11)$$

The server then replies with CK_i' to the device as part of the registration process. The device stores the value of CK_i' while the server stores the values of $\{ID_i, EXP\_Time, T_i, A_i'\}$. When the device and server authenticate one another in the next phase, these values will be necessary. The embedded device creates a l_h-bit long random number N_1 during the authentication phase. Following the calculation of P_1 and P_2 from N_1 by the following equations, the server is contacted for authentication:

$$P_1 = N_1 \times G \quad (12)$$
$$P_2 = H(P_1 \parallel N_1 \times CK_i') \quad (13)$$

Table 1. Description of the notations

Notation	Description
D_i	An embedded device registered in the system
D	The set of devices $D = \{D_1, D_2, \dots, D_1\}$
S	The server
A	The attacker
ID_1	The identification of the device D_1
H	A cryptographic hash function with an output of l_H-bit $H = \{0,1\}^* \rightarrow \{0,1\}^{l_n}$
h	A cryptographic hash function with an output of l_h-bit $h = \{0,1\}^* \rightarrow \{0,1\}^{l_h}$
Ç	An additive group implemented by an elliptic curve
G	A generator of the group $\mathcal{G}$ - a public parameter
EXP_Time	The expiry time of a particular device
X	The server's secret key
SK	A session key outputted at the end of a scheme
‖	Concatenation operation

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 13(2), JUNE, 2025
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.com.ng

Notation	Description
$\oplus$	XOR operation
$\times$	Linear multiplication with a point on the elliptic curve

The server can recompute CK using *EXP_Time*, X, and T_i and use it to determine whether P_1 and P_2 are trustworthy by computing P_2' as in the equation that follows and comparing it to P_2

$$P_2' = H(P_1 \parallel CK_i \times P_1) \quad (14)$$

Proof:

$$P_2' = H(P_1 \parallel h(SK_1) \times P_1)$$

$$P_2' = H\big(P_1 \parallel h(SK_1) \times (N_1 \times G)\big)$$

$$P_2' = H(P_1 \parallel N_1' \times G)$$

$$P_2' \equiv P_2 \quad (17)$$

The server will randomly create a long integer N_2 and calculate P_3 and P_4 with the following equations if P_2' equals P_2, which demonstrates that they are genuine.

$$P_3 = N_2 \times G \quad (18)$$

$$P_4 = H(P_2' \parallel N_2 \times A_i') \quad (15)$$

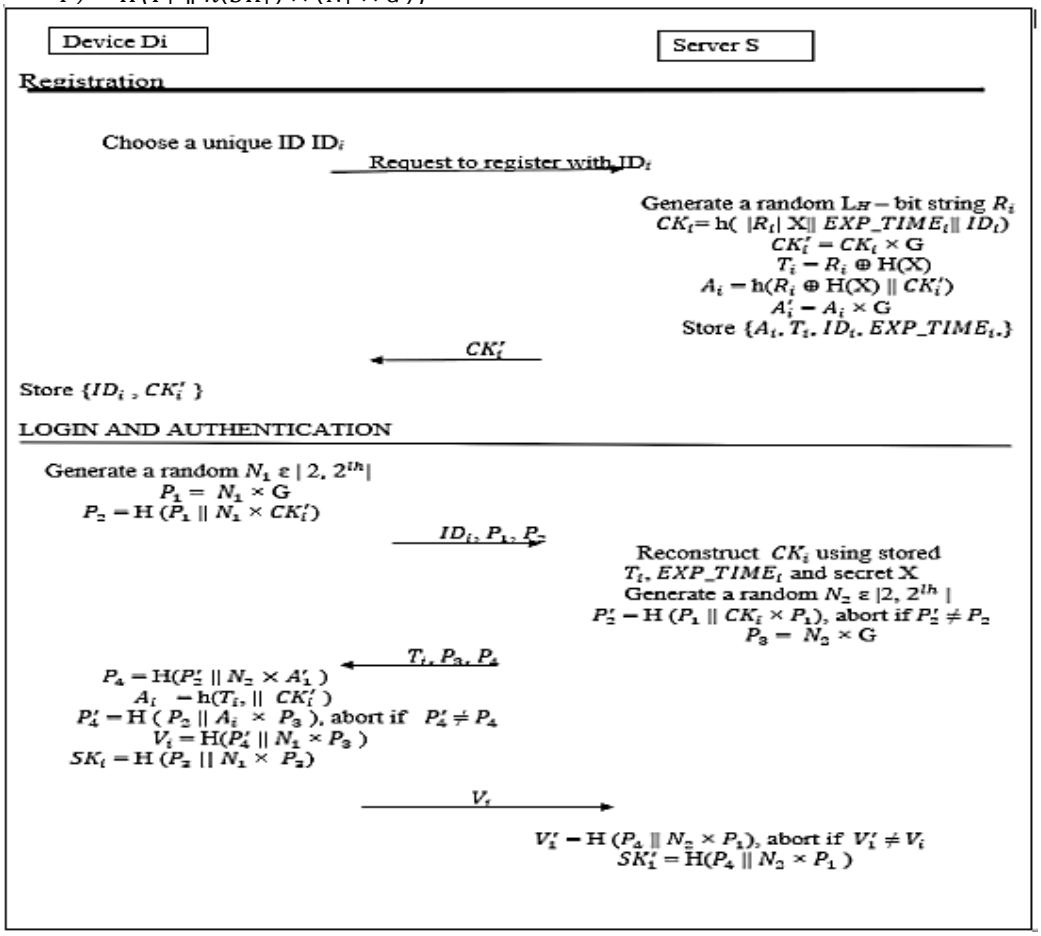

Figure 3 Authentication

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 13(2), JUNE, 2025
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.com.ng

The server gives D_i the value of $\{T_i, P_3, P_4\}$ based on the calculations. It is now the device's turn to validate the server by rebuilding A_i. Based on CK' and T_i, A_i is calculated in next equation. If A_i is accurate, then P_4' will equal P_4 in the subsequent equation. Otherwise, if P_4 and P_4' have different values, the authentication process will become unsuccessful.

$$A_i = h(T_i \times CK_i') \quad (16)$$

$$P_4' = H(P_2 \parallel A_i \times P_3) \quad (21)$$

DESIGN OF COMPRESSED SECURE DEVICE-TO-DEVICE COMMUNICATION SCHEME

In this section, we elucidate the techniques employed to develop and assess the Compressed Secure Device-to-Device Communication Scheme (CSDDCS). This protocol is crafted to facilitate the establishment of connections, authentication, and access control among IoT devices, adopting a partially centralized management approach. The SHA-1 hash function is utilized for hashing operations, and a robust elliptic curve called "brain pool" is implemented. The CSDDCS protocol encompasses two principal categories of entities, each serving distinct roles and characterized by varying resource constraints and sizes:

a) Centralized (trusted) servers: These servers are tasked with the management, authentication, storage, and regulation of device access within IoT systems. This research assumes that these servers are dependable and well-protected, making it challenging for malevolent actors to compromise their security or access sensitive data.

b) Devices: These components are under the jurisdiction of centralized servers. While devices may exhibit differences in size and capabilities, this research primarily focuses on devices with limited computational power, storage capacity, and energy resources.

Design of a Secure Device-To-Device Communication

To establish login authentication between the devices and the server, the device computes V_1 and SK_1 as in following equations using P_3 and P_4' from equations **Error! Reference source not found.** and **Error! Reference source not found.**.

On the device:

$$V_i = H(P_4' || N_1 \times P_{3_{comp}}) \quad (17)$$

$$SK_i = H(P_{3_{comp}} | \left| N_1 \times P_{3_{comp}} \right) \quad (23)$$

On the server:

$$V_i' = H(P_4 || N_1 \times P_{3_{decomp}}) \quad (24)$$

$$SK_i' = H(P_{3_{decomp}} || N_2 \times P_{1_{decomp}}) \quad (25)$$

After that, V_1 is transmitted to the server for comparison with the computed V_i' in equation **Error! Reference source not found.**. The value of V_i' should be the same as V_1. The authentication procedure will succeed if it does. Additionally, the values of SK_1 in equation **Error! Reference source not found.** and SK_i' in equation **Error! Reference source not found.** will be equal. The secret key between D_i and the server for their current session is then SK_1.

The D_1 creates a random number N_1 of l_h bits to be used in the authentication request with D_2. Then, using point compression, P_1 and P_2 are computed using N_1. Let the resulting point of the scalar multiplication of N_1 and the base point G be the following equation in order to calculate the sign bit and encode the x-coordinate:

$$N_1 \cdot G = (x_1, y_1) \quad (18)$$

$$P_1 = (x_1, y_1) \quad (27)$$

To compress the point in equation **Error! Reference source not found.**, the sign bit is determined based on the y-coordinate. If y_1 is even, the sign bit is set to 0, or if y_1 is odd, the sign bit is set to 1. Encoding the x-coordinate involves representing the x-coordinate x_1 as a binary number and to ensure the binary representation has the same length as the elliptic curve's field size. In order to create the compressed form of P_1, the sign bit and the binary representation of the x-coordinate must be concatenated. The compressed form will be referred to as P_{1_comp}.

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 13(2), JUNE, 2025
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.com.ng

$$P_{1\_comp} = \text{Compress}((N_1 \cdot G) = (x_1, y_1)) \quad (19)$$

$$P_{1\_comp} = \text{Encode}(x_1) \oplus \text{SignBit}(y_1) \quad (20)$$

In the equation that follows, $N_1{}'$ is calculated by multiplying N_1 by a scalar called h that is generated from the private key SK_1:

$$N_1' = N_1 \cdot h(SK_1) \quad (21)$$

$$P_2 = H(P_1\_comp \parallel N_1' \times G) \quad (22)$$

A connection request is sent from D_1 to D_2 along with P_{1_comp}, P_2, and ID_1. D_2creates a second random nonce with the same length as $N_1(l_{\text{h}} - bit)$in response to this request. This nonce allows D_2 to calculate P_3 as:

$$P_3 = N_2 \times G \quad (23)$$

$$P_{3\_comp} = \text{Compress}((N_2 \cdot G) = (x_3, y_3)) \quad (24)$$

$$P_{3\_comp} = \text{Encode}(x_3) \oplus \text{SignBit}(y_3) \quad (25)$$

By sending this request together with the value of P_{3_comp} to the server, D_2 then requests that it confirm D_1's connection request. A common session key is generated after each device has been authenticated by the server. As a result, all further communications between the devices and the server will be secured by encryption using the session keys to guard against listening in and tampering. $\text{Enc}()$ is used to indicate such encrypted conversations.

In order to determine if D_1 and D_2 have the necessary permissions for communication, the server first retrieves information about them from its database when it receives a message from D_2. The server should verify that the following claims are true in further detail. If the sessions of D_1 and D_2 have not yet expired and they have successfully authenticated with the server. And if D_1 and D_2 are allowed to connect to each other and interact with one another, P_1 and its associated session key SK_1 will be used by the server along with D_1 to compute $P_2{}'$. This value needs to match the P_2 received value as proven in equation **Error! Reference source not found.** :

$$P_2' = H(P_1 \parallel h(SK_1) \times P_1) \quad (35)$$

If P_2 and $P_2{}'$ do not equal each other when compared, the server will terminate the authentication procedure between the two devices. Otherwise, the two devices can then continue producing the shared cryptographic key between them by computing P_4 as in the following equation, encrypting it using SK_2, and then delivering it back to D_2.

$$P_4 = H(P_3 \parallel h(SK_1) \times P_1) \quad (36)$$

When the value is received from the server rather than an aborting message, D_2 is assured that the connection request was indeed created by D_1 itself. D_2generates an expiry time, which is EXP.Time e_{12}, P_3 and P_4 are sent to D_1, and the following equation is used to continue calculating the common session key between them.

$$SK_{12} = H(P_1 \parallel N_2 \times P_1) \quad (26)$$

But until it receives a final affirmation from D_1 that the two devices have successfully generated the identical session key independently, D_2 will not store this session key to its database. When D_1 receives the response from D_2, it must confirm that it is D_2 and not an adversary. To accomplish this, D_1 recalculates $P_4{}'$ the following equation and contrasts the outcome with the P_4 value that it obtained fromD_2. The D_2 cannot be impersonated by an attacker since only the server with access to produce P_4 (using SK_1) and securely transfer this value to D_2 may do so. As a result, D_1 may now confirmD_2's identity.

$$P_4' = H(P_3 \parallel N_1 \times G) \quad (27)$$

The shared session key SK_{21} with D_2. The SK_{21} should equal SK_{12}, as shown by the equation

$$N_1 \times P_3 \equiv N_2 \times P_1 \equiv (N_1, N_2) \times G \quad (28)$$

After obtaining the shared session key, D_1 transmits to D_2 the encrypted version of P_1 using key SK_{21}. Similar to D_2, D_1 selects EXP_Time_{21} as the expiration time. After saving I ID_2, EXP_Time_{21}, and SK_{21} to its memory, it completes the authentication process.

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

$$SK_{21} = H(P_1 \parallel N_1 \times P_3) \quad (29)$$

D_1 confirms to D_2, who then receives it. The confirmation message is then decrypted using the key generated. D_2 eventually adds ID_1, SK_{12}, and EXP_Time $_{12}$ to its storage if P_1 can be determined from the encrypted message. Because any hacker can intercept a connection request from D_1 and repeat it later, this last step is essential. D_2 may update an invalid session key and break the present secure connection with D_1 if it doesn't wait for this last confirmation. The last affirmation from D_1 will therefore aid in preventing such a replay attack.

To evaluate the compressed form P_1 and P_3 at the server in equations **Error! Reference source not found.** and **Error! Reference source not found.**, the point compression process is reversed. The sign bit and the binary representation of the x-coordinate from P_{comp} is extracted as follow:

P_{decomp} = Decompress$(P_{comp}) = (x_i, y_i)$, ∀ i = 1 or 3 (30)

P_{decomp} = Reconstruct(SignBit(P_{comp}),Decode(P_{comp})) (31)

Then the Tonelli-Shanks algorithm is performed to find the square root of $x_1 decoded\ modulo\ p$. The Tonelli-Shanks algorithm is a method used to efficiently compute the square root of a quadratic residue modulo a prime number. It was proposed by Ivan Tonelli and Daniel Shanks in the field of number theory. It computes the square root modulo a prime number, making it valuable in various cryptographic and number-theoretic applications [15]. First, the initialization of variable is set as:

$$n = p - 1 \quad (32)$$

where n represents the prime number p minus 1, which is the order of the elliptic curve group and serves as the modulus. The decompressed y-coordinate y_1, can be computed using the following expressions.

If symbol is 1, compute:

$$y_i = R \quad (33)$$

If symbol is -1, compute:

$$y_i = p - R \quad (34)$$

Here, R represents one of the square roots of $x_1 decoded\ modulo\ p$. Equations (33) and (34) represents the decompressed y-coordinate of the point P_{comp}. It is one of the two possible square roots obtained. The decompressed point becomes

$$P_{decomp} = (x, y_{decomp}) \quad (35)$$

By applying the algorithm, the compressed point P_{comp} is decompressed to obtain the y-coordinate y_{i}, of the point P_{i}. The pseudocode for the CSDDCS is given in Table 2.

Table 2: Authentication Process using Point Compression

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 13(2), JUNE, 2025
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.com.ng

```
//step 1: point compression
P1 = compress (N * G)
N1 _prime = N1 * h(SK1)
P2 = H (p1 || N1_prime *G)
// step 2: send P1, P2, and ID1 from D1 to D2
Send (p1, p2, ID1)
//step 3: Server verification
If server Verification(P1,SK1,ID1) = true, continue to step 5
Else, abort the authentication process
// step 4: Server computes p2[1] and encrypts it for D2
P2_prime = H(P1 || h(SK1) * P1)
Encrypted_P2_prime = Encrypt(P2_prime, SK2)
Server Send(D2, Encrypted_P2_prime)
// Step 5: D2 receives the message from the server
Decrypted_P2_prime = Decrypt(Encrypted_P2_prime, SK2)
P3 = Server.Verify_Confirmation(D2, Decrypted_P2_prime
// Step 6: D2 performs point decompression and final confirmation
P3_prime = Compress(P3)
DecryptedMsg = Decrypt(ReceivedMessage, SK12)
P1_prime = Extract_P1(DecryptedMsg)
DecryptedMsg = Decrypt(ReceivedMessage, SK12)
P1_prime = Extract_P1(DecryptedMsg)
Decompressed_P1 = Decompress(P1_prime)
If Decompressed_P1 == P1, continue to the next steps
Else, abort the authentication process
// Further steps for key generation and confirmation
```

The authentication process using point compression is outlined in Table 2. It begins with applying the point compression technique in Step 1, resulting in the creation of the compressed point P_1. $N_1{}'$ is then computed by scaling N_1 with a scalar derived from private key SK_1. P_2 is generated by hashing the concatenation of compressed point P_1 and is used in the authentication process. Step 2 involves D_1 sending P_1, hashed value P_2, and its identifier ID_1 to D_2 in a connection request.

Step 3 sees the server verifying D_1's connection request, checking the validity of P_1 and the authentication of D_1's ID_1. If successful, the process proceeds to Step 4, where $P_2{}'$ is computed and sent as a confirmation to D_2. The server encrypts $P_2{}'$ using D_2's session key SK_2 and sends the encrypted confirmation (Encrypted_$P_2{}'$) to D_2.

In Step 5, D_2 receives the encrypted confirmation, decrypts it, and verifies the confirmation with the server. Step 6 involves point decompression of P_3 received from the server, obtaining $P_3{}'$. D_2 decrypts the message using its session key SK_{12} shared with D_1, extracting P_1 from the decrypted message as $P_1{}'$. The D_2 decompresses $P_1{}'$ to obtain Decompressed_P_1, confirming D_1 authenticity if it matches P_1. This leads to the generation of a common session key and authentication completion. If decompressed_P_1 doesn't match $P_1{}'$, the authentication process is aborted, signaling a potential unauthorized connection.

RESULTS AND DISCUSSION

The evaluation of the scheme's performance will encompass three key metrics: communication cost, storage cost, and processing time. To assess the storage requirement, memory consumption was examined to gauge the volume of data stored on devices during the authentication process. The communication cost, indicating the volume of bytes transmitted across the communication channel throughout the entire authentication process, was calculated for all individual operations on the devices, spanning from the initialization stage to the final authentication phases, encompassing both device-to-server and device-to-device communication. The simulation is carried out using the Python 3.5 programming language on a Core-i5-3320M CPU @ 2.60GHz, 8GB RAM. The energy consumption is measured using the formula $E = V \times I \times t$, where V is the voltage (3.3 V) and I is the current (1.5 mA) of the circuit, and t is the execution time in seconds. The overall energy consumption is then calculated by multiplying the energy with the execution time, and the unit of energy consumption is in milli-Joule.

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

The communication cost is determined by the number of bits transmitted over the communication medium by the communicating devices during the authentication process, it is measure in bit. Storage requirements refer to the number of bits stored in memory to support the authentication process; it is the sum total of all the bits stored in the memory. Using the value of the current consumption for transmitting and receiving a byte extracted from the data sheet of the Tmote sky [16]. The energy consumed by a transmitting device can be calculated as:

$$E_s = \frac{\text{number of sent bits}}{\text{byte}} \times 17.4mA \quad (36)$$

$$E_r = \frac{\text{number of received bits}}{\text{byte}} \times 19.9mA \quad (37)$$

The transmission time is the total time taken for transmitting device to send one byte of data to the receiving device:

$$T_t = \frac{\text{number of sent bits}}{8} \times 0.032ms \quad (38)$$

Analysis of communication overhead

Table 3, Table 4 and Table 5 present a comparative analysis of the CSDDCS and the base scheme [1] to showcase the number of bits transmitted and received from the devices. In the base scheme of Table 3, D_1 sends various data to D_2, including identifiers (ID) which occurred three times represented by 64 bits each, points (P_1) with 512 bits each occurring three times, hashed values (P_2 and V_i) with 160 bits, and additional data. The operation of P_2 occurred two times at 160 bits each. The total number of bits sent from D_1 in the base scheme is 2208 bits. In the CSDDCS, the data sent from D_1 to D_2 has been compressed using the point compression technique.

Table 3. Bits sent from D_1

	Base scheme	**CSDDCS**
ID×3	192	192
V_i	160	160
P_1×3	1536	792
P_2×2	320	320
Total	2208(bits)	1464(bits)

This results in smaller compressed points (P_1) with 264 bits each, while the hashed values (P_2 and V_i) remain unchanged with 160 bits each. The total number of bits sent from D_1 in the CSDDCS is 1464 bits, which is significantly lower than the base scheme (2208 bits). Hence, the percentage improvement for bits sent from D_1 using CSDDCS over the base scheme is 33.70%.

Table 4. Bits sent from D_2

	Base scheme	CSDDCS
ID×3	192	192
P_1×2	1024	528
P_2×2	320	320
V_i	160	160
P_3×2	1024	528
P_4	160	160
Total	2880(bits)	1888(bits)

Similar to D_1 in Table 3, D_2 sends data to D_1 in both the base scheme and the CSDDCS in Table 4. The data includes identifiers (ID) represented by 64 bits each, compressed points (P_1 and P_3) with 512 bits each in the base scheme and 264 bits each in the CSDDCS, hashed values (P_2, V_i, and P_4) with 160 bits each, and additional data. The total number of bits sent from D_2 in the base scheme is 2880 bits. In the CSDDCS, the data sent from D_2 to D_1 has also been compressed using the point compression technique. This results in smaller compressed points (P_1 and P_3) with 264 bits each, while the hashed values (P_2, V_i, and P_4) remain unchanged with 160 bits each.

Table 5. Bits received by D_1

	Base scheme	**CSDDCS**
ck_prime	512	264
T_i	64	64
P_3	1024	528
P_4	320	320
Total	1920(bits)	1176(bits)

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

The total number of bits sent from D2 in the CSDDCS is 1888 bits, which is significantly lower than the base scheme (2880 bits). Hence, the percentage improvement for bits sent from D2 using CSDDCS over the base scheme is 34.44%. In the base scheme in Table 5, D_1 receives data from D_2, including the common cryptographic key (ck_prime) with 512 bits, intermediate variables (T_i) with 64 bits, points (P_3) with 512 bits each, hashed values (P_4) with 160 bits each, and additional data. The total number of bits received by D_1 in the base scheme is 1920 bits. In the CSDDCS, the data received by D_1 from D_2 has been compressed using the point compression technique. This results in a smaller common cryptographic key (ck_prime) with 264 bits and compressed points (P_3) with 264 bits, while the intermediate variables (Ti) and hashed values (P_4) remain unchanged with 64 bits and 160 bits, respectively. The total number of bits received by D_1 in the CSDDCS is 1176 bits, which is significantly lower than the base scheme (1920 bits). Hence, the percentage improvement for bits received by D_1 using CSDDCS over the base scheme is 38.75%.

Table 6. Bits received by D_2

Notation	Base scheme	CSDDCS
ck_prime	512	264
T_i	64	64
P_3×2	1024	528
P_4×2	320	320
P_1×2	1024	528
P_2×2	320	320
ID	64	64
Total	3168(bits)	1928(bits)

In the base scheme of Table 6, D_2 receives data from D_1, including the common cryptographic key (ck_prime) with 512 bits, intermediate variables (T_i) with 64 bits, points (P_3) with 512 bits each, hashed values (P_4) with 160 bits each, additional data, and compressed points (P_1) with 512 bits each. The total number of bits received by D_2 in the base scheme is 3168 bits. In the CSDDCS, the data received by D_2 from D_1 has been compressed using the point compression technique. This results in a smaller common cryptographic key (ck_prime) with 264 bits, intermediate variables (T_i) with 64 bits, compressed points (P_3 and P_1) with 264 bits, while the hashed values (P_4) remain unchanged with 160 bits. The total number of bits received by D_2 in the CSDDCS is 1928 bits, which is significantly lower than the base scheme (3168 bits). Hence, the percentage improvement for bits received by D_2 using CSDDCS over the base scheme is 39.14%. The result analysis and comparison of communication cost (Comm cost in bits), time (sec) and energy (micro-joules) between CSDDCS and the base scheme are presented in Table 7

Table 7: The communication cost

	Comm cost (Base)	Comm cost (CSDDCS)	Time (Base)	Time (CSDDCS)	Energy (Base)	Energy (CSDDCS)
ck_1_prime xy	512	264	2.048	1.056	1113.6	574.2
[P_1.x, P_1.y]	2560	1320	10.24	5.28	5568	2871
P_1	1024	528	4.096	2.112	2227.2	1148.4
P_2	640	640	2.56	2.56	1392	1392
P_3	1024	528	4.098	2.112	2227.2	1148.4
P_4	640	640	2.56	2.56	1392	1392

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

t_i	128	128	0.512	0.512	278.4	278.4
D_1	192	192	0.768	0.768	417.6	417.6
V_i	320	320	1.28	1.28	696	696
Total	7040 bits	4560 bits	28.16 ms	18.24 ms	15312 µJ	9918 µJ

In Table 7, the communication cost (acronym in the Table as Comm cost), time, and energy consumption for both the base scheme and the CSDDCS are presented. The CSDDCS significantly reduced the communication cost compared to the base scheme due to the compressed points. For each data element, the number of bits sent or received is reduced. The reduction ranges from 64 bits to 264 bits, depending on the data element. Overall, the CSDDCS achieved a reduction of 2480 bits in communication cost. The CSDDCS also reduced the time required for communication between devices. The time reduction is consistent for each data element, with each taking approximately half the time in the CSDDCS compared to the base scheme. The reduction is around 0.992 seconds for each data element, resulting in an overall time reduction of 9.92 seconds. Furthermore, the CSDDCS reduced energy consumption during communication. The reduction is consistent for each data element, with each consuming approximately half the energy compared to the base scheme. The reduction is around 5654 micro-joules for each data element, resulting in an overall energy reduction of 5394 micro-joules.
The CSDDCS demonstrated remarkable improvements in communication cost, time, and energy consumption as depicted in Figure 3, Figure 4 and Figure 5.

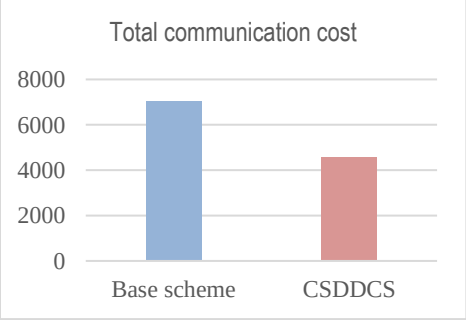

Figure 3. Total communication cost

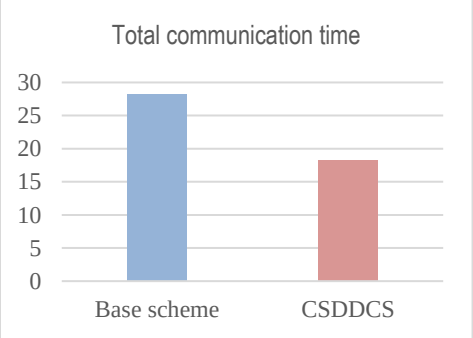

Figure 4. Total communication time

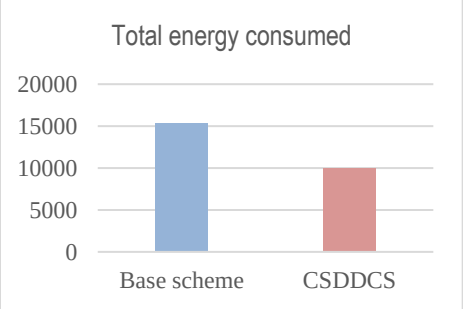

Figure 5. Total energy consumed

As shown in Figure 3, CSDDCS achieved a reduction of 2480 bits in communication cost, which is significant for secure device-to-device communication. The time reduction of 9.92 seconds and energy reduction of 5394 micro-joules indicated a more efficient and energy-saving communication process as depicted in Figure 4 and Figure 5, respectively. This shows a percentage improvement of 35.23% over the base scheme. Hence, the CSDDCS outperformed the base scheme in terms of communication cost, time, and energy consumption, making it a more efficient and practical solution for secure device-to-device communication.

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 13(2), JUNE, 2025
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.com.ng

Storage Cost Analysis

The storage cost for both the CSDDCS and the base scheme is discussed in this section to determine the overall amount of memory consumed during the whole communication session. The evaluation results are reported in Table 7 and illustrated in Figure 6.

Table 7: Result comparison of storage requirements

	Base	CSDDCS
P_1	512	264
P_2	160	160
P_3	512	264
P_4	160	160
N_1	160	160
A_i	160	160
V_i	160	160
SK	160	160
CK'	512	264
G	512	264
	3008 bits	2016 bits

In Table 7, the storage cost (bits) for both the base scheme and the CSDDCS are presented. The CSDDCS reduced the storage cost significantly compared to the base scheme. For each data element, the number of bits required for storage is reduced. The reduction ranges from 48 bits to 448 bits, depending on the data element. Overall, the CSDDCS achieved a reduction of 992 bits in storage cost. As depicted in Figure 6, the CSDDCS demonstrated considerable improvements in storage cost compared to the base scheme with percentage improvement of 32.97%. It achieved a reduction of 992 bits, which indicates more efficient utilization of storage resources for secure device-to-device communication.

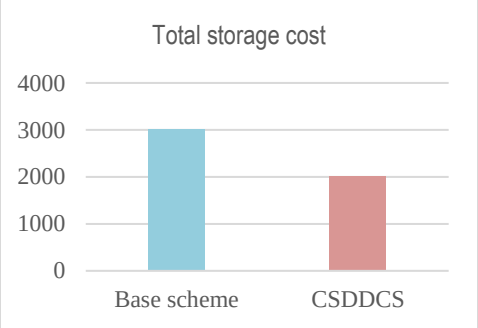

Figure 6. Total storage cost

Hence, the CSDDCS outperformed the base scheme in terms of storage cost, making it a more efficient and practical solution for secure device-to-device communication, not only in terms of communication overhead but also in storage resource utilization.

Computational cost analysis

In this section, the execution time performance of the authentication process in both the base scheme and the Compressed Secure Device-to-Device Communication Scheme (CSDDCS) is evaluated. The time taken for authentication to complete in various phases is measured, including device-to-server (D2S) authentication, device-to-device (D2D) authentication for device one (D_1) and device two (D_2), and the overall time taken for the entire authentication process from start to stop. The results of the time taken in seconds for each authentication phase are presented in Table 8, which provides an overview of the devices' processing time measured in seconds.

Table 8. Devices Processing Time

Authentication Phases	Base scheme	CSDDCS
D2S authentication	0.311321025	0.327799816
D2D for D1	0.181122389	0.183378181
D2D for D2	0.092560098	0.093169329
Full authentication	1.492600617	1.525859036

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

During the device-to-server (D2S) authentication phase, both the base scheme and the CSDDCS took approximately 0.31132s and 0.3277s, respectively, to complete the authentication. In the D2D authentication for D_1, the base scheme took about 0.18112s, while the CSDDCS took about 0.18337s. Similarly, in the D2D authentication for D_2, the base scheme took about 0.09256s, and the CSDDCS took about 0.09316s. Finally, for the full authentication phase, the base scheme required approximately 1.4926s, while the CSDDCS needed approximately 1.5258s. To better understand the comparison between the base scheme and the CSDDCS, the average processing time of 100 runs for each scheme is presented in Figure 7. The results revealed the time taken by each scheme for individual authentication phases and the overall full authentication process.

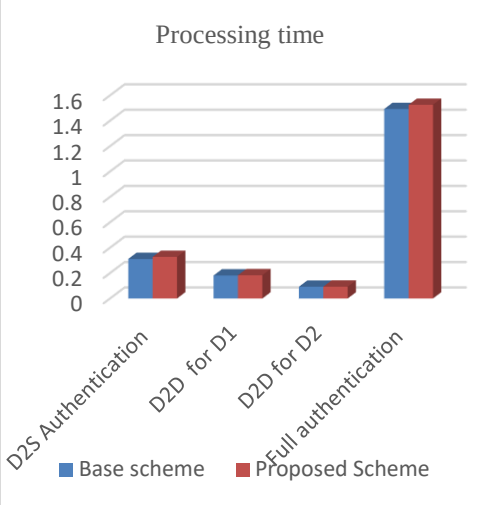

Figure 7. Processing time

From the results in both Table 8 and Figure 7, it could be noticed that the base scheme exhibited slightly better performance in terms of processing time compared to the CSDDCS, which indicated a decrease of 2.228%. However, it is important to note that this performance tradeoff is expected due to the implementation of the point compression technique in the CSDDCS, which introduced some additional executive time for the whole authentication. Despite the marginal difference, the CSDDCS still provided significant reductions in communication and storage costs, making it a valuable alternative for efficient and secure device-to-device communication.

CONCLUSION

This study introduces an enhanced energy-efficient D2D communication scheme, known as CSDDCS, to overcome the limitations associated with resource-constrained IoT devices. CSDDCS incorporates a compression mechanism within ECC to optimize resource utilization while ensuring secure authentication for both device-to-server and device-to-device communications in IoT systems. The scheme focuses on reducing communication overhead and optimizing storage resources during the authentication process. When a device gains approval to join the system, the server aids in its authentication with other devices.

By compressing ECC points, CSDDCS enhances energy efficiency without compromising security. Comparative analysis with the base scheme demonstrates significant improvements, including a notable reduction of 2480 bits in communication cost, decreased transmission time by 9.92s, and lowered energy consumption by 5394 micro-joules. Moreover, storage costs are reduced by up to 32.97%, with a decrease of 992 bits. Although there is a slight 2.228% increase in authentication completion time compared to the base scheme, the overall enhancements in communication, time, and energy consumption make CSDDCS a practical and efficient solution for secure device-to-device communication, particularly in smart cities and IoT environments. In conclusion, CSDDCS offers a promising advancement in D2D communication, delivering energy-efficient, secure, and resilient solutions for IoT settings.

REFERENCES

1. Dang, T.K., C.D. Pham, and T.L. Nguyen, *A pragmatic elliptic curve cryptography-based extension for energy-efficient device-to-device communications in smart cities.* Sustainable Cities and Society, 2020. 56: p. 102097.

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.

2. Li, N., D. Liu, and S. Nepal, *Lightweight mutual authentication for IoT and its applications.* IEEE Transactions on Sustainable Computing, 2017. 2(4): p. 359-370.
3. Wang, K.-H., et al., *A secure authentication scheme for internet of things.* Pervasive and Mobile Computing, 2017. 42: p. 15-26.
4. Afroz, T., M.N.U. Bhuiyan, and M.N. Uddin. *A Secure Mutual Authentication Protocol for IoT using ID Verifier Based on ECC*. in *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)*. 2019. IEEE.
5. Trivedi, H.S. and S.J. Patel, *Design of secure authentication protocol for dynamic user addition in distributed Internet-of-Things.* Computer Networks, 2020. 178: p. 107335.
6. Wu, H.-L., C.-C. Chang, and L.-S. Chen, *Secure and anonymous authentication scheme for the internet of things with pairing.* Pervasive and Mobile Computing, 2020. 67: p. 101177.
7. Thakare, A. and Y.-G. Kim, *Secure and efficient authentication scheme in IoT environments.* Applied Sciences, 2021. 11(3): p. 1260.
8. Hammad, M., et al., *A Provable Secure and Efficient Authentication Framework for Smart Manufacturing Industry.* IEEE Access, 2023.
9. Dasgupta, D., A. Roy, and A. Nag, *Advances in user authentication*. 2017: Springer.
10. Noori, D., H. Shakeri, and M. Niazi Torshiz, *An elliptic curve cryptosystem-based secure RFID mutual authentication for Internet of things in healthcare environment.* EURASIP Journal on Wireless Communications and Networking, 2022. 2022(1): p. 64.
11. Rostampour, S., et al., *ECCbAP: A secure ECC-based authentication protocol for IoT edge devices.* Pervasive and Mobile Computing, 2020. 67: p. 101194.
12. Wagner, L. *Basic Elliptic Curve Cryptography*. 2018; Available from: https://blog.goodaudience.com/very-basic-elliptic-curve-cryptography-16c4f6c349ed.
13. Eagle, P.N., S.D. Galbraith, and J. Ong, *Point compression for Koblitz elliptic curves.* Cryptology ePrint Archive, 2009.
14. Fan, X., et al., *Multiple point compression on elliptic curves.* Designs, Codes and Cryptography, 2017. 83: p. 565-588.
15. Sarkar, P., *Computing square roots faster than the Tonelli-Shanks/Bernstein algorithm.* Advances in Mathematics of Communications, 2022: p. 0-0.
16. Croce, S., F. Marcelloni, and M. Vecchio, *Reducing power consumption in wireless sensor networks using a novel approach to data aggregation.* The Computer Journal, 2008. 51(2): p. 227-239.
17 Momoh, M.O., Chinedu, P.U., Nwankwo, W., Aliu, D. and Shaba, M.S., 2021. Blockchain Adoption: Applications and Challenges. *International Journal of Software Engineering and Computer Systems*, *7*(2), pp.19-25.
18 Momoh, M.O., Shobowale, K.O., Abubakar, Z.M., Yahaya, B. and Ibrahim, Y., 2022. Blockchain Adoption in Aviation: Opportunities and Challenges.
19 Daniel, A., Shaba, S.M., Momoh, M.O., Chinedu, P.U. and Nwankwo, W., 2021. A computer security system for cloud computing based on encryption technique. *Computer Engineering and Applications*, *10*(1), pp.41-53.

Corresponding author: Joel Haruna
harunaj@custech.edu.ng
Computer & Electrical, Confluence University of Science and Technology, Osara, Kogi State - Nigeria.