Jurnal ELTIKOM:
Jurnal Teknik Elektro, Teknologi Informasi dan Komputer

Vol. 7, No. 2, December 2023, page. 160-169
ISSN 2598-3245 (Print), ISSN 2598-3288 (Online)
DOI: http://doi.org/10.31961/eltikom.v7i2.860
Available online at http://eltikom.poliban.ac.id

THE HYBRID CRYPTOGRAPHIC ALGORITHMS FOR SECURE RFID DATA PROTECTION IN THE INTERNET OF THINGS

Alief Vickry Thaha Maulidzart[1], Robby Kurniawan Harahap[1*], Antonius Irianto Sukowati[2], Dyah Nur'ainingsih[1], Widyastuti[1]
[1)] Department of Electrical Engineering, Universitas Gunadarma, Depok, Indonesia
[2)] Department of Electrical Engineering, Universitas Cendekia Abditama, Karawaci, Indonesia
e-mail: alief.maulidzart@gmail.com, robby_kurniawan@staff.gunadarma.ac.id, irianto@cendekia.ac.id, dyahnur@staff.gunadarma.ac.id, widyast@staff.gunadarma.ac.id

Received: 8 August 2023 – Revised: 15 November 2023 – Accepted: 17 November 2023

ABSTRACT

RFID is often used by companies to identify employees and company assets, as well as in supermarkets to identify goods when shopping. In this increasingly sophisticated era, IoT technology has wide applications. The use of RFID technology in IoT networks may pose vulnerabilities to security and privacy because it contains sensitive information, and RFID data transmitted over communication channels is vulnerable to attacks. IoT technology has characteristics such as high autonomous data capture rate, network connectivity, and interoperability for services and applications. Therefore, this research aims to improve the security of RFID data by taking into account the characteristics of IoT. The method used is hybrid cryptography by combining AES (Advanced Encryption Standard) and ECDH (Elliptic-curve Diffie-Hellman) keys. AES, as a commonly used symmetric cryptography, is chosen to protect the data, while ECDH, as the latest asymmetric cryptography, is used for a faster and more efficient process compared to previous asymmetric methods. This study utilizes the Python programming language on Jupyter Notebook. The initial step of the study involved scanning the RFID data to be secured and configuring the key on ECDH. The subsequent process included encryption and decryption of the data. The study successfully tested the success of encryption and decryption on RFID UIDs. The test data includes the result display of the hybrid encryption, the encryption and decryption processing time, and the file size of the encryption (ciphertext) and decryption (decodetext). These results show an excellent level of security for RFID UIDs. Only those with a specific key can know the contents of the cipher. It should be noted that this study was only conducted at the program level and was not implemented on hardware. Therefore, the results can be a valuable reference for future research.

Keywords*: AES, ECDH, hybrid cryptographic, internet of things, RFID.*

I. INTRODUCTION

INTERNET of Things (IoT) in general describes the specified communication among physical object to exchange data over a network communication [1]. IoT empowers objects to connect and communicate over the Internet, bridging the physical and digital worlds. In this process, data can be collected and disseminated into information systems with a high degree of granularity [2]. Companies often use RFID technology to identify employees and company assets, including its use in supermarkets to identify goods when shopping. With today's technological advancements, life is getting more and more sophisticated. In the context of IoT, the technology has various fields of use, introducing the concept of hyperconnectivity where businesses and individuals can easily communicate from far-flung locations [3]. However, the use of RFID technology in IoT networks can pose vulnerabilities to security and privacy. RFID technology contains sensitive information, and RFID data transmitted over communication channels is vulnerable to attacks. RFID is used to transmit data, so it is necessary to pay attention to security issues such as encryption, authentication, authorization, secure routing, and other data and network security [4].

This study aims to make sure the security of RFID data in order to reduce problems such as eavesdropping or brute force attacks. Therefore, cryptography was developed as a method of protecting data from external threats or leaking data confidentiality. Cryptography is described as "the science and art of encrypting messages in an incomprehensible form in order to keep them confidential" [5]. For generations, cryptography has been utilized as an encryption system. Cryptography is used in a variety of communications, from the military to commercial activity. Cryptography became important to the developing of the global economy as the internet and electronic commerce increased, and millions of individuals use it every day. Passwords, bank records, credit card statements, and personal correspondence must be encrypted or altered so that only authorized people can access them [6].

The purpose of this research is to protect RFID UID data from unexpected attacks, such as abusive attacks or eavesdropping. The Advanced Encryption Standard (AES) algorithm has been widely adopted in various encryption systems, designed to protect data and maintain privacy. Compared to algorithms such as IDES and 3DES, AES is chosen for its higher speed and security, making it an effective choice for the encryption process [7]. The Elliptic Curve Diffie-Hellman (ECDH) protocol is a key agreement scheme that allows parties A and B to create a shared secret key for use in a private key algorithm. In the information exchange stage, both parties exchange public information. The shared secret key can be generated by both using their respective public and private information. A third party, without having access to the private information of both parties, cannot calculate the shared secret key from publicly available information [8].

Research [9] identified the problem of regularly feeding and watering pets when the owner is not at home or close to the animal. Integration of feeding and drinking equipment with Internet of Things (IoT) technology was proposed as a solution to overcome this obstacle. This research creates an IoT-based system with remote control and monitoring, allowing pet owners to give food and drink through smartphone devices. The system can be operated either manually or automatically. Through this research, pet owners can leave their animals without worries as they can monitor the animals' feeding and drinking schedules through smartphones. The contribution of this research lies in the understanding that IoT can simplify and provide faster access to users. Therefore, this research serves as a reference for relevant IoT characteristics in developing such a solution.

According to research [10], the most crucial aspect in this network is the communication between smart devices. The accuracy of a device's behavior is highly dependent on its efficiency in sending data correctly. Therefore, security is very important in the implementation of the Internet of Things (IoT). Previous research has performed key exchange using Elliptic Curve Diffie-Hellman (ECDH) on the NIST P-192 curve. This research contributes as a reference that ECDH cryptography can work well for IoT devices. However, in this study, ECDH key exchange is performed using the P-256 curve to maintain data confidentiality.

In [11], the algorithm used combines the advanced encryption standard (AES) of symmetric encryption algorithms and elliptic curve encryption of asymmetric encryption algorithms. The process involves using AES to encrypt the plaintext block, followed by data compression technology to obtain the block cipher. Next, the MAC address and the AES key encrypted by Elliptic Curve Cryptography (ECC) are combined to form the complete ciphertext message. This research shows that AES cryptography can be combined with other cryptography, in this case by using ECC. The choice to use AES-128 in this research was due to the need for a smaller and faster loop in securing data. Thus, this research combines the advantages of both algorithms to achieve an optimal level of security.

In [12], Internet of Things (IoT) technology was implemented in the form of a smart home with security capabilities, such as a smart door that can open or lock the door automatically when recognizing the face of the homeowner. This research aims to improve the end-to-end data security of the model by applying the Advanced Encryption Standard (AES) algorithm. The results of the model comparison experiment show an increase in device resource requirements, which is as much as 0.81% increase in processing time, 18% increase in CPU usage, 5.3% increase in data usage, and 5.04% increase in memory usage during the process. Despite the increase in performance requirements, this study concluded that the security provided by the Advanced Encryption Standard algorithm in protecting device and server data is worth the increase. Therefore, future research is planned to use Hybrid Cryptography by utilizing AES and Elliptic Curve Diffie-Hellman (ECDH) keys as an additional step

in securing data.

According to research [13], data encryption and eavesdropping protection can be a solution to maintain data security in the cloud. The Advanced Encryption Standard (AES) algorithm is considered a faster, popular, and widely adopted choice with better computing capacity to involve symmetric encryption. Meanwhile, the asymmetric Elliptic Curve Cryptography (ECC) algorithm was chosen because it has low power consumption and more efficient computation with a minimum key length of 160 bits compared to RSA's 1024 bits, but still with the same level of security. The implementation of cryptographic algorithms is considered to address the existing Cloud vulnerabilities by ensuring secure computing. Encrypting data is considered an effective way to provide data security and privacy by preventing unauthorized access. Therefore, this research seeks a solution with a hybrid approach to data encryption, which ensures an optimal level of confidentiality. This search uses a hybrid approach method by combining AES-128 and Elliptic Curve Diffie-Hellman (ECDH) with P-256 curves as data security measures.

When Internet of Things devices connect to the internet, data security and privacy are crucial considerations. Solutions for IoT confidentiality should address high scalability requirements, heterogeneity in building blocks, and resource constraints of embedded devices, such as energy and computing constraints [14]. AES and ECC are the finest symmetric and asymmetric encryption algorithms, respectively, according to the given answer. The algorithm's lack of complexity in terms of algorithm code and ciphertext makes it the most appealing option for cryptography specialists. Small and compact algorithms, such as ECC and AES, can utilize fewer resources and provide greater security than other cryptography [15].

This research combines the Advanced Encryption Standard (AES) and elliptic-curve cryptography (ECC) algorithms with the objective of securing hybrid cryptography algorithms for protecting RFID confidential data. RFID data is encrypted and described using AES, a symmetric algorithm, and ECC, an asymmetric algorithm defined as ECDH. With three distinct key lengths of 128 bits, 192 bits, and 256 bits, as well as a packet size of 128 bits, the symmetric AES encryption systems in this group are extremely versatile. Consequently, AES encryption technology is widely implemented in hardware and software [16]. Elliptic Curve Diffie Hellman is a variant of Diffie Hellman. This algorithm generates keys using elliptic curves [17]. Under the master agreement scheme, all parties involved in a particular communication are required to provide some form of data or information for the creation of the shared session key [18]. This research begins with RFID data collection outside of the initial program, followed by testing to encrypt and characterize RFID data, describing encryption results, and describing the encryption process's duration and decryption. The benefit of this research is that when there is Radio-Frequency Identification (RFID) connected to the Internet of Things (IoT), the data from the Unique Identifier (UID) can be kept confidential using hybrid cryptography. The UID data encryption process in this context is considered very fast, short, and small in size.

II. RESEARCH METHOD

This research implements on Jupyter Notebook and the Python programming language. AES (Advanced Encryption Standard) is the most efficient symmetric encryption algorithm widely used in IT industries lately. AES is a popular encryption method due to its high level of security. It will be effective, simple, and supported on virtually all platforms. AES supports various key lengths, including 128, 192, and 256 bits. However, in this investigation, a 128-bit key and 16 iterations were used. Meanwhile, Elliptic Curve Cryptography (ECC), as a newer form of public key cryptography, provides advantages in fast key agreement, fast signatures, and efficient key generation in practice. While ECC does not provide an explicit encryption mechanism, this research adopts a hybrid encryption scheme with the Elliptic Curve Diffie-Hellman (ECDH) key exchange scheme. This scheme facilitates the acquisition of public keys for ECC encryption and decryption processes, allowing two parties to generate public-private key pairs for elliptic curves over an insecure channel.

Figure 1 shows the flowchart with the hybrid encryption and decryption process. The first stage of preprocessing before encryption is to scan various RFID tag data using Arduino Uno and RFID modules as its attachment components. Next, the configuration key is prepared to be matched with the Elliptic Curve Diffie-Hellman (ECDH) key to encrypt and describe the information contained in the RFID data.

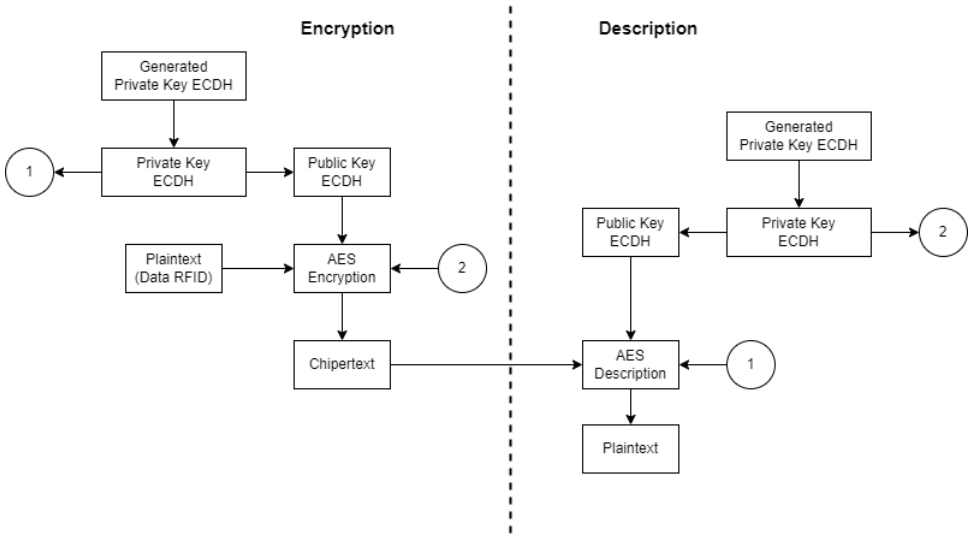

Figure 1. Flowchart Encryption and Descripton Hybrid

```
------------------------------------------------------------
encryption Key (ECDH): 697831462505796649579499205152035665873002698782338187339985812442062651678 33
------------------------------------------------------------
```

Figure 2. ECDH Encryption Key result for first attempt encryption

```
------------------------------------------------------------
encryption Key (ECDH): 5806389780075294343626359581332292189566811092527032676081794942127124810027
------------------------------------------------------------
```

Figure 3. ECDH Encryption Key result for second attempt encryption

The encryption program in ECDH requires the public key and secret key to generate the ciphertext, while the decryption program in ECDH requires the public key and secret key to describe the result of the ciphertext.

The ECDH rule allows you to substitute the values of alicePrivKeyG and bobPrivKeyG over an insecure channel if you have two secret integers, a and b (the private keys that identify Alice and Bob), and an ECC elliptic curve with a Generator (G) point. The result, (1) uses Alice and Bob's public keys.

$$alicePubKey \times bobPrivKey = bobPubKey \times alicePrivKey \quad (1)$$

Some standard processing rounds involve four functional steps that modify each common AES data block. There are four steps in each round of data encryption, called SubBytes, ShiftRows, MixColumns, and AddRoundKey. Each small square in the graph represents one byte, or 8 bits, and each matrix has a total of 16 or 128 bit small squares. After successful encryption, the encryption result is stored in a file.

Next, the same hybrid decryption process is performed using the key from ECDH. The decryption results will be stored in a file. The last stage is the analysis stage, where after successful encryption and decryption, AES-ECDH is analyzed using parameters such as encryption output, encryption time, decryption time, and storage space to be encrypted and decrypted.

III. Result and Discussion

Research on a hybrid cryptographic algorithm based on AES and ECDH has been completed. This process followed the relevant NIST 800-22 Rev 1a guidelines regarding Pseudorandom Number Generators. This stage includes RFID data collection, hybrid encryption algorithm test results, hybrid decryption, hybrid encryption processing time, hybrid decryption time, encrypted document file size, and correlation between plaintext and encryption results. In this study, two tests were conducted, namely encryption and decryption. Before running encryption and decryption, the first step is to configure the ECDH key. Every time the algorithm is started, the private key in ECDH keeps changing. Therefore, in order to maintain harmonization between encryption and decryption, the configuration of the private key must be configured automatically in the algorithm.

Figures 2, 3, 4, and 5 show the results of automatic key generation by the ECDH algorithm that will be used as the key in the modified AES. The key is used at the beginning of each encryption and

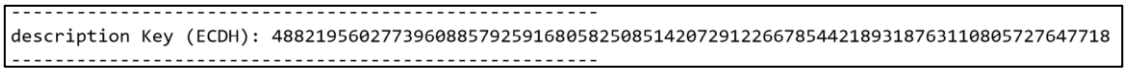

Figure 4. ECDH decryption Key result for first attempt decryption

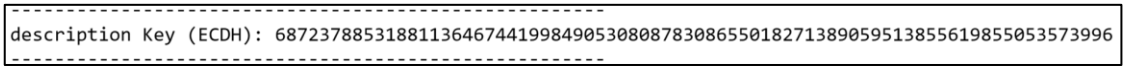

Figure 5. ECDH Decryption Key result for second attempt decryption

decryption algorithm run.

$$curve = registry.get\_curve('brainpoolP256r1') \quad (2)$$

$$AlicePrivKey = secrets.randbelow(curve.field.n)$$
$$AlicePubKey = AlicePrivKey * curve.g \quad (3)$$

Equation (2) for generating curves using the *"brainpoolP256r1"* library refers to the specific elliptic curves and associated domain parameters selected and recommended in "RFC 5639 - Brainpool Elliptic Curve Cryptography (ECC) Standard Curves and Curve Generation." It is important to note that each curve generation process may result in different parameters.

To configure ECDH encryption, a private key and a public key are required. The process of generating the private key involves the use of the *"secrets.randbelow"* module, which is a cryptographically secure and robust integral number generator (see (3)). This module is used to generate random numbers that are secure and suitable for security-sensitive applications. Next, the result of the generated *"AlicePriveKey"* is multiplied by *"curve.g"* to form a curve and generate a public key. Thus, the ECDH key configuration is considered complete after these steps are performed. Figures 2 and 3 show the ECDH key generation results for the encryption process in the first experiment and the ECDH generation results in the second experiment. The ECDH curve used in this process has a key length of 256 bits. The ECDH key serves as an identifier for the process in the next explanation.

$$bobPrivKey = secrets.randbelow(curve.field.n)$$
$$bobPubKey = bobPrivKey * curve.g \quad (4)$$

Likewise, configuring an ECDH decryption requires a private key and a public key. The process of generating the private key involves using the *"secrets.randbelow"* module as shown in (4). The result generated from *"BobPriveKey"* is multiplied by *"curve.g"* to form a curve and generate the public key. Thus, the ECDH key configuration is considered complete. After obtaining the ECDH key configuration, the process will generate the same "SharedKey" key between the encryption and decryption processes.

Figures 4 and 5 show the ECDH key generation results for the decryption process in the first experiment and the ECDH generation results in the second experiment. The ECDH curve used in both processes has a key length of 256 bits, corresponding to that used in the encryption process. The function of this ECDH key is to verify the ECDH key in the encryption process. If the results of the ECDH key (SharedKey) in the encryption and decryption processes are the same, then the decryption process will output results that match the plaintext, and vice versa. If not, the result will be different, indicating that there is a problem or failure in the decryption process.

$$bobSharedKey = alicePrivKey * bobPubKey$$
$$cipher\_key = compress(bobSharedKey) \quad (5)$$

To generate the shared key for the encryption process, the variable *"alicePrivKey"* is required to be multiplied by *"bobPubKey"* as shown in (5). After that, the shared key can be obtained. When the shared key in the encryption process is the same as the shared key in the decryption process, then chipper_key will proceed to the encryption process in AES that has been modified with ECDH.

$$aliceSharedKey = bobPrivKey * alicePubKey$$
$$cipher\_key = compress(aliceSharedKey) \quad (6)$$

Conversely, to generate the shared key in the decryption process, the variable *"bobPrivKey"* is required to be multiplied by *"alicePubKey"* as shown in (6). After that, the shared key can be obtained. When the shared key in the encryption process is the same as the shared key in the decryption process,

TABLE 1
RESULT FROM HYBRID ENCRYPTION & DECRYPTION FOR THE FIRST EXPERIMENT

Name RFID	PlainTexts	CipherTexts	Encryption Time (ms)	Decryption Time (ms)
Card RFID 1	638FE118	[VýŸõôðÈÈ▯▯ÏÈËÊ	86	68
Card RFID 2	1499C649	[VòØõÿðüË▯▯Ï▯ËÊ▯	95	72
Card RFID 3	A8972327	Ü½¨▯▯E▯▯ÏÈ▯▯▯▯Ï▯	106	81
Card RFID 4	83139205	õV§▯▯▯©üÏÈ▯▯▯▯Ï▯	98	86
Card RFID 5	E3670B04	▯G▯▯▯ÇÏÈ▯▯▯▯Ï▯	101	85
Key RFID 1	A7C6C128	ÜÀÃRðEü▯ÏÈ▯▯▯▯Ï▯	85	90
Key RFID 2	4C832949	▯Úж▯ÿÎð▯ÏÈ▯▯▯▯Ï▯	94	88
Key RFID 3	B61D62AF	A▯§▯▯▯▯^ÏÈ▯▯▯▯Ï▯	61	66
Total encryption time and decryption time			726	636

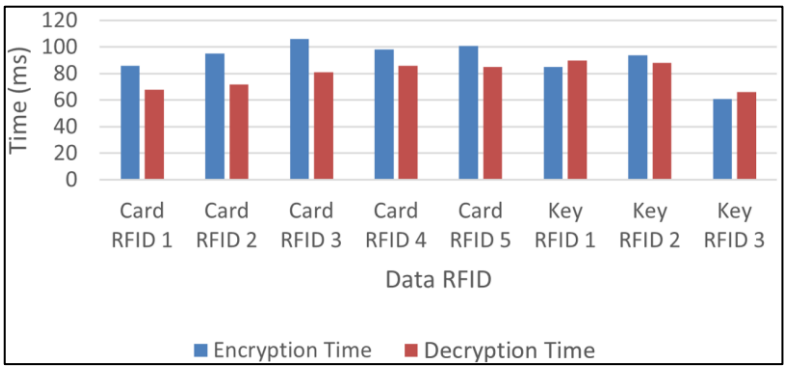

Figure 6. The difference between encryption and decryption processing time for first experiment

```
"brainpoolP256r1" => y^2 = x^3 +
566981876053261100436272283961783460771206145394
475214109386828188763884139993x +
175772324973218388410756977897945202629504260589
923084567046852300633325438902 (mod
768849563970453442208097466290016490930379502009
943055203735601445031516197751)
```

Figure 7. The encryption results in data values

```
768849563970453442208097466290016490927375317844145295387555190630635363590
79 *
(6324372974956233335529224355031297033477817557105472658709538162362714411
4786,
382186150937535238931222779640308103875854055397726025815578318874857179975
975) on "brainpoolP256r1" => y^2 = x^3 +
566981876053261100436272283961783460771206145394752141093868281887638841399
93x +
175772324973218388410756977897945202629504260589230845670468523006333254389
02 (mod
768849563970453442208097466290016490930379502009430552037356014450315161975
51)
                    public key: (After Get Compress)
0x1b6712ed97718849c7486e6967ae74878b62b6514d0695c6a3a1b1130d5bf7540
```

Figure 8. The result of the ECDH private key is multiplied by "*curve.g*" to generate the ECDH public key

then chipper_key will proceed to the decryption process in AES that has been modified with ECDH. Therefore, the test data taken involves two experiments. In the first experiment, 8 different RFID UIDs (RFID Card and RFID Key) were used, while for the second experiment, 10 different UIDs (RFID Card) were used. The test data includes displaying the encryption result (Ciphertext), encryption and decryption processing time, file size of the original file, ciphertext file, and decodetext file.

Table 1 and Figure 6 show the encryption and decryption results of the first experiment. This experiment was conducted 8 times, involving 5 RFID cards and 3 RFID keychains. In the first stage, "RFID Card 1" with UID 638FE118 was encrypted using the curve generated by the "brainpoolP256r1" library. The encryption results in data values as shown in Figure 7.

To generate the ECDH Private Key for the encryption process, the "*secrets.randbelow*" module is used. The result produces the following data: `768849563970453442208097466290016490927375`

```
def subBytes(A):
  s_box = np.load('Lookup Tables/s_box.npy')
  B = np.zeros((4,4),dtype=int)
  for row in range(4):
    for col in range(4):
      sub_row, sub_col = A[row,col]//16, A[row,col]%16
      B[row,col] = s_box[sub_row,sub_col]
  return B
```

Figure 9. SubBytes block stage

```
def shiftRows(A):
  B = np.zeros((4,4),dtype=int)
  # keep 1st row intact
  B[0,:] = A[0,:]
  # shift each element of 2nd row 1 step to the left
  B[1,0],B[1,1],B[1,2],B[1,3] = A[1,1],A[1,2],A[1,3],A[1,0]
  # shift each element of 3rd row 2 steps to the left
  B[2,0],B[2,1],B[2,2],B[2,3] = A[2,2],A[2,3],A[2,0],A[2,1]
  # shift each element of 4th row 3 steps to the left
  B[3,0],B[3,1],B[3,2],B[3,3] = A[3,3],A[3,0],A[3,1],A[3,2]
  return B
```

Figure 10. ShiftRow process

```
def mixCol(A):
  e_table = np.load('Lookup Tables/E_Table.npy')
  B = np.zeros((4,4),dtype=int)
  for row in range(4):
    for col in range(4):
      sub_row , sub_col = A[row,col]//16,A[row,col]%16
      B[row,col] = e_table[sub_row,sub_col]
  return B
```

Figure 11. MixColumn Process

```
def addRoundKey(A,key):
    B = np.zeros((4,4),dtype=int)
    B = np.bitwise_xor(A,key)
    return B
```

Figure 12. Result Matrix of MixColumn Process

`3178441452953875551906306353635907 9`. Then, the result of the ECDH private key is multiplied by *"curve.g"* to generate the ECDH public key to be used is shown in Figure 8. After obtaining the private key and public key, shared key generation is performed, where the shared key from the encryption and decryption process must have the same value. The shared key result for "RFID Card 1" is: `0x8c3252aa74fabc0ab074fb66f26d6825864336b4df4e34f98ed599859be01c001`.

Once the ECDH keys are equalized, the next process involves the use of AES. In the first block, the text on the RFID UID is converted to Unicode (matrix). Next, using the S-box table as a guide for the Substitution operation, the SubBytes block stage is performed with the data derived from the first block conversion procedure in Figure 9. Next, by applying a shift, the ShiftRow process is performed on the data blocks that have gone through the SubBytes stage. This process is executed on 3 rows of the matrix as shown in Figure 10.

Next, proceed to the next block stage, which is MixColumn. This process gives a diffusion effect to the ciphertext, where each column is treated as a 4-sided polynomial. For simplicity, use the "L table" help table (see Figure 11).

The next process is addRoundKey, where a bitwise XOR operation is performed between a round key and the result matrix of the MixColumn (see Figure 12).

Finally, the result of the AES process is converted back from Unicode (matrix) to text. This process is repeated from the first block to the last block in the AES process, a total of 16 rounds. The decryption process in AES, like the encryption process, is also done in reverse order. First, the text is converted to Unicode (matrix), then addRoundKey, InvMixColumn, InvShiftRow, InvSubBytes are removed, and finally converted back to text.

All ciphertexts are the result of the encryption process in the first experiment using the ECDH key (Figure 2). Based on Table 1, the fastest encryption time in the first experiment was achieved by "RFID Key 3" at "61 ms" with UID "B61D62AF", while the longest encryption time was performed by "RFID Card 3" at "106 ms" with UID "A8972327", with a total encryption time of 636 ms. In this first experiment, it can be observed that the longest time in decryption is "90 ms" belonging to "RFID Key

TABLE 2
RESULT FROM HYBRID ENCRYPTION & DECRYPTION FOR THE SECOND EXPERIMENT

Name RFID	PlainTexts	CipherTexts	Encryption Time (ms)	Decryption Time (ms)
Card RFID 6	F3F2BADF	▯V▯▯Ù▯▯F▯▯▯Ê▯▯▯È	65	60
Card RFID 7	0DD60D04	Â×▯X▯Æ▯Å▯▯▯Ê▯▯▯È	41	50
Card RFID 8	AB95B9DF	Ü þ£ü▯▯F▯▯▯Ê▯▯▯È	60	71
Card RFID 9	41C755FF	▯¾▯▯¥▯▯§▯▯▯Ê▯▯▯È	69	101
Card RFID 10	979ABBDF	ùÀþßD▯▯F▯▯▯Ê▯▯▯È	75	75
Card RFID 11	271ABADF	▯Àñ ßÙ▯▯F▯▯▯Ê▯▯▯È	70	40
Card RFID 12	D9CDBBDF	▯±▯▯D▯▯F▯▯▯Ê▯▯▯È	70	55
Card RFID 13	A7E6B8DF	ÜÀ¨Xð▯▯F▯▯▯Ê▯▯▯È	80	65
Card RFID 14	00A8BADF	Â▯Ûöù▯▯F▯▯▯Ê▯▯▯È	59	60
Card RFID 15	22F7BBDF	▯^▯▯D▯▯F▯▯▯Ê▯▯▯È	80	73
Total encryption time and decryption time			669	650

TABLE 3
RESULT SIZE FROM HYBRID ENCRYPTION & DECRYPTION FOR THE FIRST EXPERIMENT

Name RFID	Original File Size	Encryption File Size	Decryption File Size
Card RFID 1	7 Bytes	30 Bytes	16 Bytes
Card RFID 2	8 Bytes	31 Bytes	16 Bytes
Card RFID 3	8 Bytes	31 Bytes	16 Bytes
Card RFID 4	8 Bytes	29 Bytes	16 Bytes
Card RFID 5	8 Bytes	30 Bytes	16 Bytes
Key RFID 1	7 Bytes	30 Bytes	16 Bytes
Key RFID 2	7 Bytes	30 Bytes	16 Bytes
Key RFID 3	8 Bytes	28 Bytes	16 Bytes
Average Size	7.625 Bytes	29.875 Bytes	16 Bytes

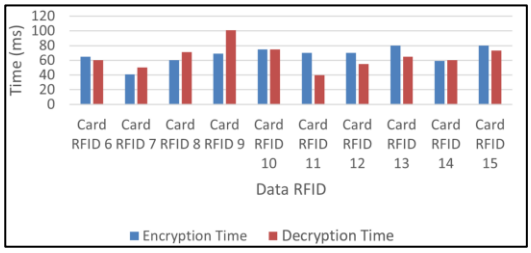

Figure 13. The difference between encryption and decryption processing time for Second experiment

1" with UID "A7C6C128", while the fastest decryption time is "66 ms" belonging to "RFID Key 3" with UID "B61D62AF", with a total decryption time of 636 ms.

Figure 6 shows the time difference between the encryption and decryption results in the first experiment. It can be seen that the encryption processing time is longer than the decryption time. The blue bar line shows the encryption time, while the red bar line shows the decryption time. The average encryption time is 90.75 ms, while the average decryption time is 79.5 ms.

In Table 2 and Figure 13, the results of the second study were conducted on 10 different RFID UIDs. It can be seen in Table 2 that the fastest encryption time in the first test was achieved by "RFID Card 7" with a time of "41 ms" and UID "0DD60D04", while the longest encryption time was performed by "RFID Card 13" with UID "A7E6B8DF" and "RFID Card 15" with UID "22F7BBDF", both reaching a time of "80 ms". The total encryption time was 636 ms. Then, in this second experiment, it can be seen that the longest decryption time is "101 ms" for "RFID Card 9" with UID "41C755FF", while the fastest decryption time is "66 ms" for "RFID Card 11" with UID "271ABADF". The total decryption time was 636 ms.

In Figure 13, we can see the time difference between the encryption and decryption results in the second experiment. The blue bar line shows the encryption time, while the red bar line shows the decryption time. It can be seen that the encryption process time and the decryption process time do not have a significant difference with a relatively small time difference. If calculated, the average encryption time is 66.9 ms, while the average decryption time is 65 ms.

Table 3 and Figure 14 show the file sizes of plaintext, ciphertext, and decodetext from the first experiment. Based on Table 3, the smallest plaintext file size is 7 bytes for 3 RFID files, while the largest is 8 bytes for 5 RFID files, with an average total size of 7.625 bytes. Furthermore, for the encrypted file

TABLE 4
RESULT SIZE FROM HYBRID ENCRYPTION & DECRYPTION FOR THE SECOND EXPERIMENT

Name RFID	Size File original	Size File Encryption	Size File Decryption
Card RFID 6	8 Bytes	29 Bytes	16 Bytes
Card RFID 7	8 Bytes	31 Bytes	16 Bytes
Card RFID 8	7 Bytes	30 Bytes	16 Bytes
Card RFID 9	7 Bytes	32 Bytes	16 Bytes
Card RFID 10	8 Bytes	30 Bytes	16 Bytes
Card RFID 11	7 Bytes	30 Bytes	16 Bytes
Card RFID 12	7 Bytes	30 Bytes	16 Bytes
Card RFID 13	8 Bytes	30 Bytes	16 Bytes
Card RFID 14	8 Bytes	31 Bytes	16 Bytes
Card RFID 15	7 Bytes	28 Bytes	16 Bytes
Average Size	7,5 Bytes	30,1 Bytes	16 Bytes

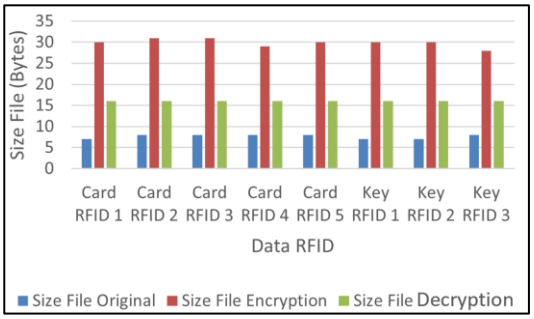

Figure 14. Size difference between plaintext, ciphertext, decodetext for first experiment

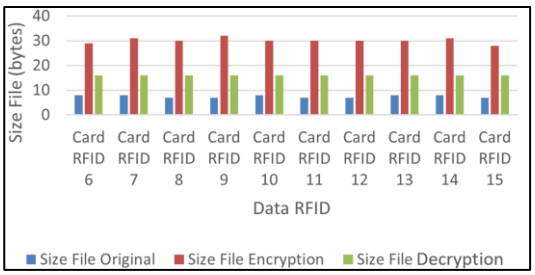

Figure 15. Size difference between plaintext, ciphertext, decodetext for first experiment

size (ciphertext), the largest was 31 bytes belonging to "RFID Card 2" and "RFID Card 3", while the smallest was 28 bytes belonging to "RFID Key 3", with an average size of 29.875 bytes. For the decryption size results (decodetext), all files on RFID have a size of 16 bytes.

Figure 8 describes the size comparison between the original file, the encrypted file (Ciphertext), and the decrypted file (Decodetext) from the first experiment. The blue bar line represents the original file, the red bar line represents the encrypted file (Ciphertext), and the green bar line represents the decrypted file (Decodetext). Based on these three variables, the largest overall size is found in the encrypted file because there are many characters that are difficult to read by humans and are random. Meanwhile, the smallest overall size is found in the original file. In the second experiment, the average size difference between the original file and the encrypted file was 22.25 bytes.

Table 4 and Figure 15 show the plaintext, ciphertext, and decodetext file sizes of the second experiment. Based on Table 4, the smallest plaintext file size is 7 bytes for 3 RFID files, and the largest is 8 bytes for 5 RFID files, with an average total size of 7.5 bytes. Furthermore, for the encrypted file size (ciphertext), the largest is 31 bytes belonging to "RFID Card 7" and "RFID Card 14", while the smallest is 29 bytes belonging to "RFID Card 6", with an average size of 30.1 bytes. For the decryption size results (decodetext), all files on RFID have a size of 16 bytes.

Figure 15 describes the size comparison between the original file, the encrypted file (Ciphertext), and the decrypted file (Decodetext) from the second experiment. The blue bar line represents the original file, the red bar line represents the encrypted file (Ciphertext), and the green bar line represents the decrypted file (Decodetext). Based on these three variables, the largest overall size is found in the encrypted file because there are many characters that are difficult to r[Decryption]ns and are random. Meanwhile, the smallest overall size is found in the original file. In the second experiment, the average

size difference between the original file and the encrypted file was 22.6 bytes.

IV. CONCLUSION

IoT features high-level autonomous data acquisition, network connectivity, and interoperability of services and applications. RFID technology in IoT networks often poses a danger to security and privacy. Due to the confidential nature of the data transmitted over communication channels, they are vulnerable to attacks. The solution to this research problem uses hybrid cryptography to protect the privacy of RFID data. The hybrid cryptography procedure used in this research is AES with ECDH key. The results of encryption and decryption trials were successfully carried out, with encryption done manually, and decryption done by manually entering the encrypted file results. This research is in accordance with the guidelines in NIST 800-22 Rev 1a. For the future, it is suggested that this research can be implemented on hardware using microcontrollers. This research can be a reference for researchers who conduct additional research, and can be developed by including an intuitive interface.

REFENCES

[1] S. Gabsi, Y. Kortli, V. Beroulle, Y. Kieffer, A. Alasiry, and B. Hamdi, "Novel ECC-based RFID mutual authentication protocol for emerging IoT applications," *IEEE access*, vol. 9, pp. 130895–130913, 2021.
[2] W. Viriyasitavat, T. Anuphaptrirong, and D. Hoonsopon, "When blockchain meets Internet of Things: Characteristics, challenges, and business opportunities," *J. Ind. Inf. Integr.*, vol. 15, pp. 21–28, 2019.
[3] L. Tawalbeh, F. Muheidat, M. Tawalbeh, and M. Quwaider, "IoT Privacy and security: Challenges and solutions," *Appl. Sci.*, vol. 10, no. 12, p. 4102, 2020.
[4] J. R. Naif, G. H. Abdul-Majeed, and A. K. Farhan, "Secure IOT system based on chaos-modified lightweight AES," in *2019 International Conference on Advanced Science and Engineering (ICOASE)*, IEEE, 2019, pp. 1–6.
[5] R. Munir, "Kriptografi Edisi Kedua," *Bandung. Penerbit Inform.*, 2019.
[6] R. H. Prayitno, S. A. Sudiro, S. Madenda, and S. Harmanto, "Hardware Implementation Of Galois Field Multiplication For Mixcolumn And Inversemixcolumn Process In Encryption-Decryption Algorithms," *J. Theor. Appl. Inf. Technol.*, vol. 100, no. 14, 2022.
[7] T. Hidayat and R. Mahardiko, "A Systematic literature review method on aes algorithm for data sharing encryption on cloud computing," *Int. J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 49–57, 2020.
[8] S. Aikins-Bekoe and J. Ben Hayfron-Acquah, "Elliptic curve diffie-hellman (ECDH) analogy for secured wireless sensor networks," *Int. J. Comput. Appl.*, vol. 176, no. 10, pp. 1–8, 2020.
[9] R. K. Harahap, E. P. Wibowo, D. Nur'ainingsih, A. K. Wijaya, and R. A. S. C. Anindya, "Dogs Feed Smart System With Food Scales Indicator IoT Based," in *2022 4th International Conference on Cybernetics and Intelligent System (ICORIS)*, IEEE, 2022, pp. 1–7.
[10] R. K. Kodali and A. Naikoti, "ECDH based security model for IoT using ESP8266," in *2016 International conference on control, instrumentation, communication and computational technologies (ICCICCT)*, IEEE, 2016, pp. 629–633.
[11] T. Yue, C. Wang, and Z. Zhu, "Hybrid encryption algorithm based on wireless sensor networks," in *2019 IEEE international conference on mechatronics and automation (ICMA)*, IEEE, 2019, pp. 690–694.
[12] W. Adhiwibowo, A. M. Hirzan, and M. S. Suprayogi, "Peningkatan Keamanan Data End-to-End Smart Door Menggunakan Advanced Encryption Standard," *J. ELTIKOM J. Tek. Elektro, Teknol. Inf. dan Komput.*, vol. 6, no. 2, pp. 186–194, 2022.
[13] A. Orobosade, T. A. Favour-Bethy, A. B. Kayode, and A. J. Gabriel, "Cloud application security using hybrid encryption," *Commun. Appl. Electron.*, vol. 7, no. 33, pp. 25–31, 2020.
[14] P. M. Chanal and M. S. Kakkasageri, "Hybrid algorithm for data confidentiality in Internet of Things," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, IEEE, 2019, pp. 1–5.
[15] S. Farooq and P. Chawla, "A novel approach of asymmetric key generation in symmetric AES via ECDH," *Int. J. Syst. Assur. Eng. Manag.*, vol. 11, no. 5, pp. 962–971, 2020.
[16] R. H. Prayitno, S. A. Sudiro, and S. Madenda, "Avoiding Lookup Table in AES Algorithm," in *2021 Sixth International Conference on Informatics and Computing (ICIC)*, IEEE, 2021, pp. 1–6.
[17] L. Widyawati, H. Husain, M. Azwar, and M. C. S. Girsang, "Analisa Perbandingan Hybrid Cryptography RSA-AES dan ECDH-AES untuk Keamanan Pesan," *J. Teknol. Inf. dan Komput.*, vol. 9, no. 2, 2023.
[18] G. Kanda, A. O. A. Antwi, and K. Ryoo, "Hardware architecture design of AES cryptosystem with 163-bit elliptic curve," in *Advanced Multimedia and Ubiquitous Engineering: MUE/FutureTech 2018 12*, Springer, 2019, pp. 423–429.