



## Article

# Dynamic Key Replacement Mechanism for Lightweight Internet of Things Microcontrollers to Resist Side-Channel Attacks

Chung-Wei Kuo , Wei Wei, Chun-Chang Lin, Yu-Yi Hong, Jia-Ruei Liu and Kuo-Yu Tsai \*

Department of Information Engineering and Computer Science, Feng-Chia University, Taichung City 407, Taiwan; cwkuo@fcu.edu.tw (C.-W.K.); weiking1021@gmail.com (W.W.); m1205003@o365.fcu.edu.tw (C.-C.L.); m1221097@o365.fcu.edu.tw (Y.-Y.H.); d0909002@o365.fcu.edu.tw (J.-R.L.)

\* Correspondence: kytsai@o365.fcu.edu.tw

**Abstract:** 5G technology and IoT devices are improving efficiency and quality of life across many sectors. IoT devices are often used in open environments where they handle sensitive data. This makes them vulnerable to side-channel attacks (SCAs), where attackers can intercept and analyze the electromagnetic signals emitted by microcontroller units (MCUs) to expose encryption keys and compromise sensitive data. To address this critical vulnerability, this study proposes a novel dynamic key replacement mechanism specifically designed for lightweight IoT microcontrollers. The mechanism integrates Moving Target Defense (MTD) with a lightweight Diffie–Hellman (D-H) key exchange protocol and AES-128 encryption to provide robust protection against SCAs. Unlike traditional approaches, the proposed mechanism dynamically updates encryption keys during each cryptographic cycle, effectively mitigating the risk of key reuse—a primary vulnerability exploited in SCAs. The lightweight D-H key exchange ensures that even resource-constrained IoT devices can securely perform key exchanges without significant computational overhead. Experimental results demonstrate the practicality and security of the proposed mechanism, achieving key updates with minimal time overhead, ranging from 12 to 50 milliseconds per encryption transmission. Moreover, the approach shows strong resilience against template attacks, with only two out of sixteen AES-128 subkeys compromised after 20,000 attack attempts—a notable improvement over existing countermeasures. The key innovation of this study lies in the seamless integration of MTD with lightweight cryptographic protocols, striking a balance between security and performance. This dynamic key replacement mechanism offers an effective, scalable, and resource-efficient solution for IoT applications, particularly in scenarios that demand robust protection against SCAs and low-latency performance.

**Keywords:** 5G; Internet of Things (IoT); side-channel attack (SCA); microcontroller unit (MCU); Diffie–Hellman (D-H); moving target defense (MTD); AES-128



Academic Editor: Francesco Buccafurri

Received: 27 November 2024

Revised: 10 January 2025

Accepted: 15 January 2025

Published: 18 January 2025

**Citation:** Kuo, C.-W.; Wei, W.; Lin, C.-C.; Hong, Y.-Y.; Liu, J.-R.; Tsai, K.-Y.

Dynamic Key Replacement Mechanism for Lightweight Internet of Things Microcontrollers to Resist Side-Channel Attacks. *Future Internet* 2025, 17, 43. <https://doi.org/10.3390/fi17010043>

**Copyright:** © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

As the Internet of Things (IoT) and wireless communication continue to grow, they transform people’s lives through progressive technologies, creating new opportunities for development and enhancing both quality of life and productivity. IoT has also made modern infrastructures more interconnected and optimized. Statista [1] noted there will be 4 billion connected IoT devices by the end of 2024. This growth shows that more and more people are using IoT to make their lives and work easier. Some devices use Wi-Fi to connect and send data, allowing users to control IoT devices by only one smartphone. However, as IoT devices become more common, cybersecurity becomes a serious issue. Many devices send sensitive data over public networks, making them easy to steal. Insecure

data transmission not only compromises personal privacy but also exposes vulnerabilities in essential systems that rely on IoT infrastructure. Strong cryptographic algorithms need to be used when transmitting data.

Common encryption methods like the Advanced Encryption Standard (AES) and RSA are keeping data safe. They make it hard for attackers to break encrypted data easily. The cryptographic strength of AES and RSA imposes resource-intensive demands on attackers attempting to break the encryption. However, SCAs pose a significant threat by bypassing the mathematical security of these algorithms. SCAs exploit the physical characteristics emitted by encryption devices, such as power consumption [2], electromagnetic radiation [3], and thermal emissions [4], to infer encryption keys. These physical leakages often exhibit statistical correlations with the keys, allowing attackers to deduce them through sophisticated analysis.

Current SCA defense strategies primarily focus on microcontrollers, utilizing techniques such as masking [5] and hiding [6] to minimize the observability of physical signals, thereby improving resistance to SCAs. Yet, as SCA techniques continue to evolve in complexity, traditional defenses have proven insufficient [7]. Furthermore, the rapid progress in quantum computing has introduced additional threats to existing security mechanisms. Consequently, the development of more robust defense strategies has become imperative.

Recently, researchers have proposed an SCA method targeting Wi-Fi environments, which enables attackers to decrypt transmitted data without the need for access to the device's password [8]. This study highlights the significant threat that SCAs pose to modern information security. To address this challenge, the present research introduces a lightweight AES-128-based encryption key protection mechanism specifically designed for securing IoT communications. This mechanism effectively defends against SCAs without compromising system performance. In IoT devices, AES-128 has become the best encryption algorithm candidate due to its low computational overhead and power consumption [9]. Compared to other encryption methods, AES-128 not only provides sufficient encryption strength [10], but also operates with high efficiency. However, just relying on the encryption algorithm itself is not enough to ensure robust security. Especially in public network environments, IoT devices performing encryption operations may leak physical signals, which attackers can exploit to compromise encryption keys, thus threatening the overall security of the system. If smart home devices, such as access control or surveillance systems, fall victim to SCAs, personal privacy and sensitive data could be severely compromised [11]. Therefore, improving encryption key protection mechanisms for IoT devices is both a critical and urgent task. Furthermore, Zeng et al. [12] have proposed an enhanced Message Authentication Encryption (MAE) scheme based on the Physical Layer Key Generation (PKG), which improves encryption efficiency by 80.5% while consuming fewer computational resources. This advancement offers a more efficient security solution for resource-constrained IoT devices.

This paper proposes a dynamic AES key replacement mechanism based on the concept of MTD, coupled with the D-H key exchange protocol. This combined approach enables the dynamic replacement of AES encryption keys during transmission, significantly reducing the risk of key compromise. To evaluate the effectiveness of this approach, we constructed an IoT transmission system using an Arduino UNO and the ESP8266 Wi-Fi module [13]. During testing, attackers used probes to capture the physical traces emitted by the encryption devices and conducted statistical analysis on the GPU computing platform. Results showed that after analyzing a sufficient number of traces, the attackers were able to extract the AES encryption key. This finding highlights that static AES keys were insufficient for ensuring the security of IoT devices in public environments. Dynamic key replacement is necessary to mitigate the risk of key leakage.

To address these vulnerabilities, this study proposes a high-efficiency dynamic key replacement mechanism based on the Diffie–Hellman (D-H) protocol. Building on the Moving Target Defense (MTD) approach introduced by Vuppala et al. [14], the proposed mechanism dynamically updates encryption keys after each transmission, effectively mitigating vulnerabilities related to side-channel attacks (SCAs). Designed as a lightweight solution, this mechanism is tailored to the computational limitations of resource-constrained IoT microcontrollers, ensuring low power consumption and real-time performance. By enhancing resilience against SCAs, the proposed approach significantly reduces the risk of data leakage in IoT devices deployed in open and unsecured environments. Experimental analysis validates the mechanism’s feasibility and effectiveness, demonstrating minimal time overhead (12 to 50 milliseconds) and strong resistance to key extraction attacks, even after 20,000 attempts.

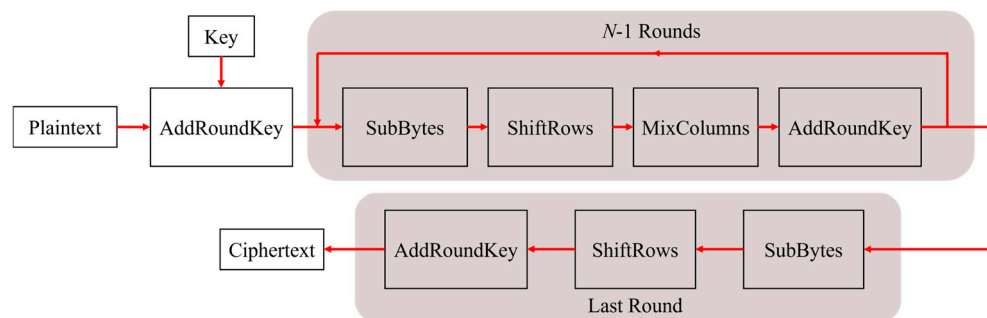
The remainder of this paper is organized as follows: Section 2 (Preliminaries) provides an overview of foundational concepts, including SCA vulnerabilities and the cryptographic mechanisms that underpin this study. Section 3 (Related Work) reviews existing research on SCA countermeasures and lightweight cryptographic protocols in IoT environments, identifying key gaps that this study addresses. Section 4 (Research Methodology) details the proposed dynamic key replacement mechanism, focusing on its integration with the D-H protocol and its lightweight implementation for IoT devices. Section 5 (Experimental Results) presents the experimental setup and findings, evaluating the mechanism’s performance and security under realistic IoT scenarios. Finally, Section 6 (Conclusions) summarizes the key contributions of this study and highlights its practical implications.

## 2. Preliminaries

### 2.1. Advanced Encryption Standard (AES)

AES was first introduced in 1998 [15,16] and officially adopted as the standard for symmetric encryption algorithms in 2001, replacing the legacy Data Encryption Standard (DES), which supported only 56-bit keys. Due to its high computational efficiency and strong security features, AES has become one of the most widely used encryption algorithms, particularly within the IoT domain. AES encryption process consists of four main stages. First, the plaintext is arranged into a  $4 \times 4$  matrix, where each element represents 1 byte. Then, the key matrix is XORed with the plaintext matrix during `AddRoundKey()`. Next, `SubBytes()` replace each byte using the S-Box lookup table to introduce nonlinear transformation to enhance security. Next, the `ShiftRows()` operation shifts the rows of the state matrix to increase data diffusion. Except in the last round, the `MixColumns()` operation is performed, which combines the four bytes in each column to obscure the relationship between plaintext and ciphertext through matrix multiplication in the Galois Field ( $GF(2^8)$ ).

The main difference between AES-128, AES-192, and AES-256 is the length of the encryption key, which impacts the number of encryption rounds; N of 10, 12 or 14 offers a flexible balance between security and performance. Given the resource limitations in IoT devices, AES-128 is widely adopted due to its low computational overhead and power consumption while still offering strong security. In AES-128, the 128-bit key is divided into 16 individual bytes, each ranging from hexadecimal 00 to FF. The encryption process, as illustrated in Figure 1, ensures that even with the shortest key length, AES provides a high level of security and performance, making it particularly well-suited for lightweight IoT systems.



**Figure 1.** AES encryption process flow.

In 2023, NIST chose ASCON [17] as the next-generation lightweight cryptographic standard for IoT encryption. ASCON was not the fastest, but it was chosen due to its security, good operation on low-power platforms, and inability to be attacked in other ways. ASCON is well-suited for resource-constrained environments. It is based on AEAD and SHA-256, providing efficient protection for short-lived, time-sensitive data commonly found in IoT ecosystems by low-power overheads. However, ASCON is not necessarily more secure than AES-128 in all cases. ASCON is best for short-lived data in low-power environments. AES-128 is still the best choice for securing persistent and sensitive data in many IoT applications. Most microcontrollers have already used well-known cryptographic algorithms like AES, RSA, ECDSA, and SHA-256. This makes AES-128 a better choice than ASCON. This study improves the AES-128 encryption algorithm for use in real-world IoT applications. Because it is compatible with many devices and provides security, improving AES-128 can benefit IoT devices that need to be secure and perform well. By refining the algorithm, we aimed to provide a solution that ensure robust protection without using too many resources.

## 2.2. SCAs

SCAs are a non-invasive technique aimed at exploiting the physical characteristics leaked by cryptographic implementations, specifically targeting the hardware behavior of microcontrollers in IoT devices. Since encryption algorithms are ultimately executed on microcontroller chips, attackers can analyze physical signals, such as power consumption or electromagnetic emissions, to infer the encryption keys. Common SCA techniques consist of Simple Power Analysis (SPA) [18], Differential Power Analysis (DPA), and Correlation Power Analysis (CPA) [19]. In this study, CPA is employed as the primary attack method because it effectively analyzes the correlation between the power consumption of the encryption device and the encryption key.

During the AES encryption process, attackers typically focus on the output of the S-Box in the first round as the Point of Interest (PoI), since at this stage, the signal has only passed through the AddRoundKey() and SubBytes() operations. Attackers record the power consumption or electromagnetic emissions during the encryption process, converting these signals into traces, which are binary representations of the recorded data. These traces are then subjected to statistical analysis.

CPA works by correlating hypothetical key guesses with the actual recorded traces, using correlation analysis to infer the correct key. Since there is a relationship between the electromagnetic emissions and power consumption, CPA groups the traces based on power consumption patterns. The power model used for grouping is the Hamming Weight (HW) [20], which categorizes each trace based on the number of '1s' in the binary representation. For AES-128, HW values range from 0 to 8, creating a total of nine groups, as shown in Figure 2 [21]. By this method, the intermediate value  $x$  (the output of the first round S-Box) is obtained for a set of random plaintexts and hypothetical keys. The

traces  $y$ , recorded during the encryption process, are then used in correlation coefficient calculations to predict the correct key for the Device under Test (DUT) based on the results of the correlation analysis, as shown in Equation (1).

$$r_{j,k,t} = \frac{\sum_{i=1}^n (x_{j,k,i} - \bar{x}_{j,k})(y_{t,i} - \bar{y}_t)}{\sqrt{\sum_{i=1}^n (x_{j,k,i} - \bar{x}_{j,k})^2 \sum_{i=1}^n (y_{t,i} - \bar{y}_t)^2}} \quad (1)$$

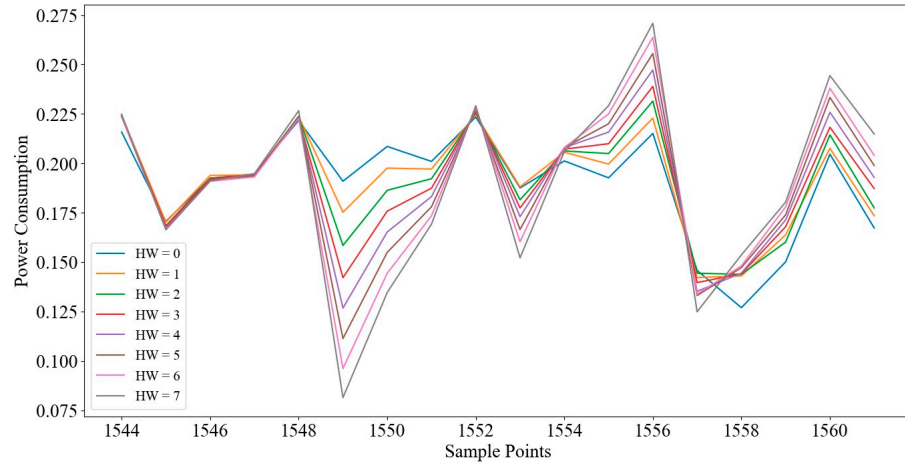


Figure 2. Hamming Weight grouping results.

### 2.3. D-H Key Exchange Mechanism

The D-H key exchange mechanism was introduced in 1976 [22] and has become a fundamental milestone in modern cryptography. D-H is one of the pioneering protocols in public key cryptography, enabling two parties to securely establish a shared symmetric key without sharing a prior key over an insecure channel. Many communication protocols utilize D-H to establish a secure channel by exchanging information over a public network. There are several variations of D-H, including the Elliptic Curve D-H Key Exchange (ECDH). Given the hardware limitations in IoT environments, the D-H implementation in this study was chosen for its minimal power consumption, ensuring compatibility with power-limited IoT devices.

The lightweight version of D-H employed in this study is based on the discrete logarithm problem, mathematically expressed as  $f(x) = G^x \text{ mod } P$ , where  $G$  is the base and  $P$  is a prime modulus.

- Private Key Generation: The IoT device and the server independently select their private keys, denoted as  $a$  and  $b$ , respectively. These private keys are large random integers chosen securely to ensure robustness against attacks.
- Public Key Calculation: Using the private keys, the respective public keys are computed as follows:

$$\text{For the IoT device: } PK_A = G^a \text{ mod } P$$

$$\text{For the server: } PK_B = G^b \text{ mod } P$$

These public keys are derived from the exponential function over a finite field and are securely exchanged over an insecure channel.

- Shared Key Derivation: Upon receiving the other party's public key, the IoT device and the server calculate the shared symmetric key  $SK$  as follows:

$$\text{For the IoT device: } SK = PK_B^a \text{ mod } P$$

$$\text{For the server: } SK = PK_A^b \text{ mod } P$$

Due to the commutative property of modular exponentiation, both calculations yield the same shared secret key  $SK$ , which forms the basis of secure communication.

As illustrated in Figure 3, in order to derive the final shared key  $SK$  from the public values, an attacker would need to solve for the private keys  $PK_A$  or  $PK_B$ , which is computationally infeasible without access to those private keys. D-H ensures strong security by relying on the exchange of public keys and the difficulty of solving the discrete logarithm problem, even over an insecure channel. This mechanism is leveraged in our study to implement dynamic key replacement, providing protection against SCAs.

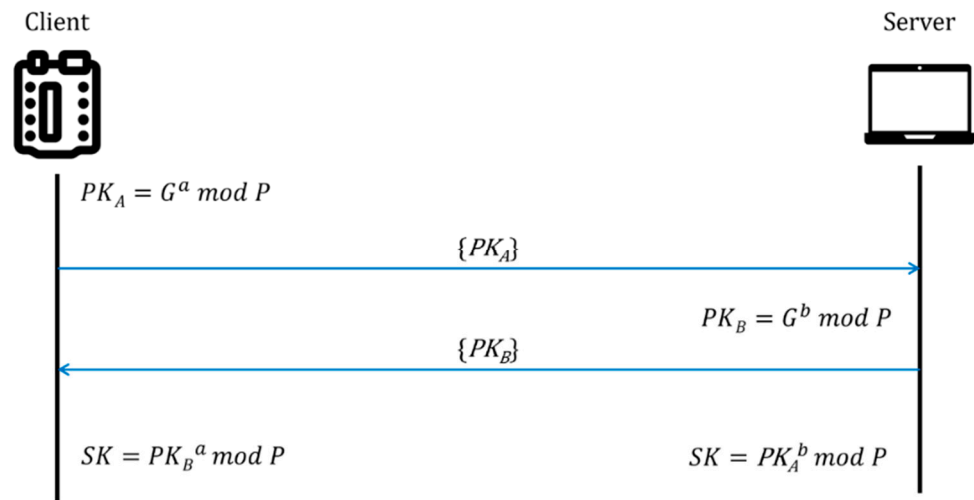


Figure 3. Lightweight D-H protocol.

#### 2.4. Threat Model

This study adopts a well-defined threat model specifically tailored to the resource-constrained nature of IoT environments and the physical characteristics of cryptographic implementations:

- **Attacker Capabilities:** The attacker is assumed to have access to non-invasive tools capable of capturing physical leakages, such as electromagnetic emissions or power consumption traces, generated during the AES encryption process. The primary attack technique considered is Correlation Power Analysis (CPA), which statistically correlates these traces with hypothetical encryption keys to deduce the correct key.
- **Attacker Limitations:** The attacker is restricted to non-invasive methods and cannot physically tamper with the microcontroller or IoT device. Additionally, the attacker has no access to the device’s internal state and relies solely on external observations of emitted signals. The attack is further constrained by real-time requirements, limiting the feasibility of collecting an excessive number of traces in practical operational environments.
- **Assumptions:** The initial key exchange between the IoT device and the server is conducted securely, ensuring the integrity of the shared key.

The devices operate in environments where physical access is limited but not entirely restricted, and the cryptographic hardware remains uncompromised during operation. This threat model serves as the foundation for assessing the proposed mechanism’s resilience against SCAs. By dynamically updating encryption keys after each cryptographic cycle, the proposed solution effectively reduces the risk of key recovery through CPA or similar statistical attacks.

### 2.5. Common Wireless Communication Protocols in the IoT

There are many types of wireless communication protocol for IoT devices. The most common are Wi-Fi, Bluetooth, Zigbee, and LoRa [23–26]. Each has different features and benefits. Table 1 compares each of the protocol’s property and advantages.

**Table 1.** Comparison of specifications of common wireless communication protocols.

Protocol	Frequency (GHz)	Data Transmission Speed (Max)	Communication Distance	Power Consumption	Advantages
Wi-Fi	2.4/5	1 Gbps	50–100 m	High	High-frequency bandwidth, high speed, supported by extensive infrastructure
Bluetooth	2.4	3 Mbps	10–100 m	Low	Low power consumption, suitable for small devices with long-term operation
Zigbee	2.4	250 kbps	10–100 m	Low	Low power consumption, supports multi-node mesh networks
LoRa	Sub-GHz	50 kbps	2–15 km	Ultra-low	Long-range coverage, ultra-low power consumption

Wi-Fi is the most pervasive and reliable protocol mentioned above, offering high data transmission rates, so it is the best candidate for smart home environments. Due to its widespread used, Wi-Fi has become the most common wireless communication protocol. Furthermore, many existing devices and infrastructures have been optimized and integrated to support it. In smart home scenarios, large amounts of real-time data transmission and many multimedia applications use communication protocols with high bandwidth and reliability. Despite the vulnerability of Wi-Fi to network attacks, particularly when handling sensitive personal data in open environments, this is one of the main focuses of this research. The aim of this study is to enhance the security of data transmitted over Wi-Fi and mitigate the risks posed by SCA by improving encryption key protection mechanisms.

### 2.6. Glossary of Acronyms

To ensure clarity and consistency, this paper adopts several commonly used technical acronyms. These acronyms are listed and defined in Table 2 below. Throughout the remainder of this paper, the abbreviations provided in the table will replace their full forms for brevity. This ensures efficient communication while maintaining readability for a technical audience.

**Table 2.** Glossary of acronyms.

Acronym	Full Form
5G	Fifth-Generation Mobile Network
AES	Advanced Encryption Standard
CPA	Correlation Power Analysis
D-H	Diffie–Hellman
IoT	Internet of Things
MAC	Message Authentication Code
MITM	Man-in-the-Middle
MTD	Moving Target Defense
RSA	Rivest–Shamir–Adleman Encryption
SCA	Side-Channel Attack
Wi-Fi	Wireless Fidelity

### 3. Related Work

SCAs represent a significant threat to the security of IoT devices, leveraging physical leakages such as power consumption, electromagnetic emissions, and timing information to extract sensitive cryptographic keys. Addressing these vulnerabilities has been a major focus in both academic and industrial research, leading to the development of hardware- and software-based countermeasures.

#### 3.1. Hardware-Based Countermeasures

Hardware-based countermeasures aim to reduce physical leakage or obscure the correlation between cryptographic operations and observable signals. Techniques such as masking and hiding are widely employed:

- **Masking:** Introduces random noise to intermediate computations, breaking the statistical dependency between leaked signals and secret keys, as demonstrated by Mangard et al. [27].
- **Hiding:** Utilizes techniques such as dynamic voltage and frequency scaling (DVFS) to reduce the signal-to-noise ratio of the leakage, effectively obscuring attack vectors [18].

Recent advancements, such as those proposed by Zhao et al. [28], focus on integrating multiple defense layers into chip-level designs. Their On-Chip Monitoring (OCM) circuits enable real-time detection of attack attempts, while Backside Buried Metal (BBM) structures provide enhanced electromagnetic shielding to reduce leakage. While these approaches significantly improve resistance to SCAs and fault injection attacks, they also increase fabrication complexity and costs, limiting their practicality for low-cost IoT devices.

#### 3.2. Software-Based Countermeasures

Software-based solutions focus on enhancing cryptographic algorithms and protocols to improve resistance against SCAs without extensive hardware requirements:

- **Dynamic Key Replacement:** Mechanisms such as those proposed by Vuppala et al. [14] use MTD to periodically update cryptographic keys, reducing the risk of key reuse and subsequent SCAs.
- **Lightweight Cryptographic Algorithms:** Algorithms like PRESENT and SPECK are designed to minimize computational overhead, making them suitable for IoT applications [29]. However, their security against advanced SCAs, such as template attacks, remains a topic of ongoing research.

These solutions provide flexibility and scalability but often involve trade-offs between performance and security, particularly in real-time applications with stringent resource constraints.

#### 3.3. Comparison with Prior Research

The proposed mechanism in this study builds on prior research by integrating Moving Target Defense (MTD) with a lightweight Diffie–Hellman (D-H) protocol and AES-128 encryption. This combination enables dynamic key updates after each transmission, providing robust resistance against side-channel attacks (SCAs) while maintaining low computational overhead. Unlike hardware-intensive approaches or static software solutions, the proposed method strikes a balance between security and resource efficiency, making it particularly well-suited for IoT environments.



Table 3 offers a comprehensive comparison between the proposed mechanism and selected prior works, highlighting key differences in methodologies, features, and performance. The table demonstrates how the proposed solution surpasses traditional masking and hardware-based approaches by delivering strong SCA resistance with minimal resource consumption. Additionally, compared to existing dynamic key replacement mechanisms, the proposed method introduces a more granular, per-transmission key update process, further enhancing security without compromising low-latency performance. This comparison underscores the novelty and practicality of the proposed approach in addressing the security challenges faced by resource-constrained IoT devices.

**Table 3.** Comparison of the proposed mechanism with selected prior works.

Ref.	Method	Key Feature	Overhead	SCA Resistance
[3]	Backside Buried Metal	Real-time attack detection	High (fabrication cost)	Strong for EM and fault attacks
[4]	Dynamic Key Replacement	Periodic key updates using MTD	Moderate	Limited by update frequency
[27]	Masking	Adds random noise to computations	High (hardware mods)	Strong for power analysis
[30]	Key Simplification + D-Box Updates	Reduced encryption cycles and dynamic key updates	Low	Resilient to replay attacks

## 4. Research Methodology

### 4.1. Experimental Framework

In network environments, sensitive information must be encrypted before transmission to prevent unauthorized access. In resource-constrained IoT systems, AES encryption has been widely used. While AES provides robust security, the encryption operations are executed by microcontrollers within the devices, which may leak physical signals during the encryption process. These signals could be exploited by attackers to extract the encryption keys, thus compromising the encrypted messages. To counter SCA on IoT devices, we proposed a dynamic key replacement AES encryption algorithm based on the MTD [14] concept. This approach changes the initial encryption key before an SCA can succeed.

To validate the effectiveness of this defense mechanism, we implemented a wireless communication platform for a smart access control system, which included an RFID-based access control system (Arduino UNO), a Wi-Fi module (ESP8266), and access control software. A laptop executed access control software as client and server in the IoT environment, handling communication between the devices. The smart application system was set up using actual RFID tags, with an RFID reader connected to Arduino UNO. When the RFID tags were read, the data were encrypted by AES and transmitted via the Wi-Fi module to the access control software for decryption.

Once the encryption and decryption process of the access control data was validated, we conducted an SCA on the system. Using an oscilloscope, we captured electromagnetic radiation emitted during the execution of encryption operations on Arduino UNO, specifically targeting the power consumption data. The oscilloscope used an electromagnetic probe to capture the traces, which were sent to the control computer. These traces were processed by a GPU computing platform using CPA to determine the success of the key extraction by analyzing the number of successfully attacked subkeys. The experimental setup is illustrated in Figure 4.

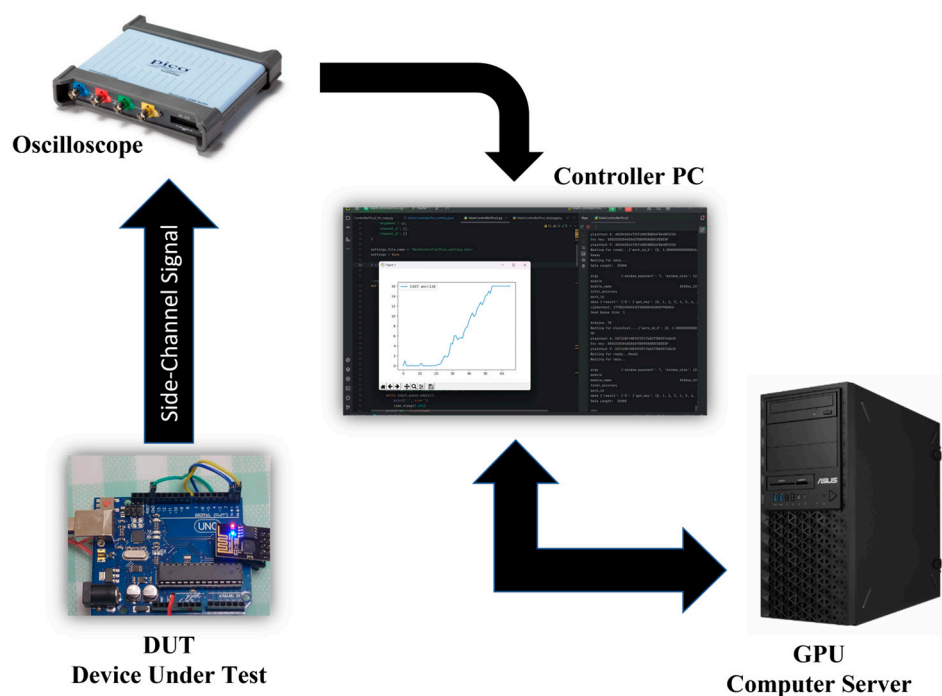


Figure 4. SCA system configuration [16].

Once we confirmed the IoT transmission scenario and SCA setup, we modified the AES encryption algorithm to enable dynamic key replacement. To evaluate the efficiency and effectiveness of the key replacement, we performed the key replacement after capturing different amounts of traces and observed the frequency limitations of the key replacement. The configuration of the client, server, and the dynamic key replacement AES design will be discussed detail in the following sections.

#### 4.2. Client-Side Setup

The communication scenario is built around the wireless transmission of access control card data using Wi-Fi. We developed a simple IoT access control system using the Arduino UNO NodeMCU v3, the ESP8266 Wi-Fi module, and the MFRC522 RFID sensor module. The client-side setup and operation are as follows:

First, we include the header files `ESP8266WiFi.h`, `MFRC522.h`, and `config.h`. The first two libraries correspond to the Wi-Fi Module and the RFID Reader, while `config.h` defines the Wi-Fi SSID, password, default AES-128 encryption key, server IP, port, and the base and modulus values for the D-H key exchange.

Next, the `setup()` function is used to initialize the MFRC522 module, connect to the specified Wi-Fi network, and establish communication with the server. The `loop()` function continuously monitors the MFRC522 sensor for access card UID readings, records the number of data packets transmitted, and encrypts the UID string using a custom AES-128 function `aes128()`. Then the encrypted data are sent to the server for processing.

To ensure transmission accuracy, the number of packets recorded by the client are compared with the number received by the server. After a certain number of transmissions, the client initiates a key negotiation with the server using the D-H protocol to generate the next AES encryption key. The actual client-side setup is illustrated in Figure 5.

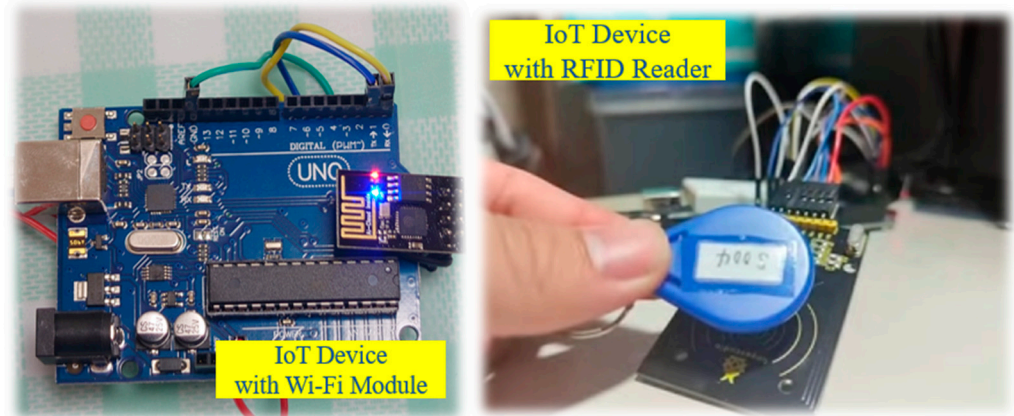
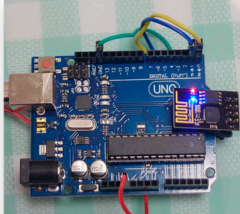


Figure 5. Actual client-side setup.

### 4.3. Server-Side Setup

The server-side program for the smart access control system was developed in Java, as illustrated in Figure 6. The program utilizes the following libraries: info.picocli:picocli, javax.crypto, and java.net. After starting the server through the command line with the specified port parameter, the java.net.ServerSocket is used to listen on the designated port. Once a communication channel is established, the server connects to the client and begins exchanging data. During the connection, the D-H key exchange is used to generate the next key from the initial key. The messages received from the client (Arduino) are decrypted using Cipher.getInstance(“AES/ECB/NoPadding”). If the message is in RFID format, the system searches for the corresponding UID and triggers specific actions, such as sending HTTP requests or logging the access time into a database. After receiving a certain number of messages, the server re-initiates the D-H key exchange with the client to generate the next AES-128 decryption key.



IoT Device with WiFi Module


```

[05-31-2023 12:44:52] Message (0)
[05-31-2023 12:44:52] Use key to decrypt: B1 9C C6 D9 75 67 2D F6 9B 30 E6 8E 77 1D 3D 6A
[05-31-2023 12:44:52] F808AFA0182371C7C0D73B40F656B4A8 --> TIMB: 20003
[TIMB System] 20003 ms from system boot

[05-31-2023 12:44:58] Message (1)
[05-31-2023 12:44:58] Use key to decrypt: ED 23 57 86 DD B2 51 84 BD 82 88 27 0F 57 D5 21
[05-31-2023 12:44:58] 816BBF5B03C44B0B64B56403599F703A --> RFID: 624AAA29
[RFID System] RFID is 624AAA29
[RFID System] S004 , welcome!

[05-31-2023 12:45:06] Message (2)
[05-31-2023 12:45:06] Use key to decrypt: DB 19 9B 1D 58 BB 2D 86 81 74 C1 5B 05 D8 B7 B1
[05-31-2023 12:45:06] C874D425E27C8A4C91DB49FB73717B4F --> RFID: 83508BD9
[RFID System] RFID is 83508BD9
[RFID System] S024 , welcome!
                    
```

Source	Destination	Protocol	Length	Info	Hex
192.168.0.107	224.0.0.1	IGMPv2	60	Membership Report group 2	0000 00 d8 61 10 d4 a9 60 01 94 22 15 eb 08 00 45 00
192.168.0.107	192.168.0.102	TCP	60	47858 -> 13579 [SYN] Seq=6	0010 00 38 02 c4 00 00 80 06 b5 da c0 a8 00 6b c0 a8
192.168.0.107	192.168.0.102	TCP	60	47858 -> 13579 [ACK] Seq=1	0020 00 66 ba f2 35 0b 00 10 9d 06 13 33 6b 65 50 18
192.168.0.107	192.168.0.102	TCP	70	47858 -> 13579 [PSH, ACK]	0030 0b 68 42 87 00 00 c8 74 d4 25 e2 7c 8a 4c 91 d0
192.168.0.107	192.168.0.102	TCP	70	47858 -> 13579 [PSH, ACK]	0040 49 fe 73 71 7b 4f



IoT Device with RFID Reader

Wireshark Software at Control Host

AES-128 Encrypted RFID Identification Code

Figure 6. Server-side configuration display.

### 4.4. Dynamic AES Key Replacement Mechanism

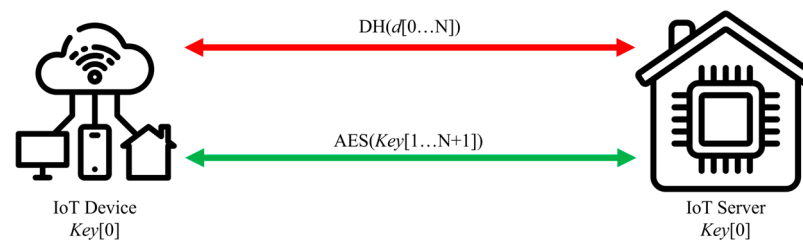
The proposed mechanism addresses the critical vulnerability of key reuse in AES encryption, which can result in power consumption or electromagnetic signal leakage that can be exploited by SCAs. This vulnerability stems from the repeated use of the same encryption key rather than the key length itself. Although longer keys, such as AES-

256, could theoretically reduce these risks, their higher computational overhead renders them impractical for resource-constrained IoT devices. Therefore, this study focuses on implementing a dynamic key replacement mechanism that enhances security without imposing additional computational burdens.

- Initial Key Configuration and Key Replacement

As shown in Figure 7, the mechanism begins with a pre-configured initial key,  $Key[0]$ , used solely during the initial connection negotiation. The D-H key exchange protocol then generates the first 32-bit hopping operator,  $d[0]$ , which is combined with  $Key[0]$ , using a hash function to derive the next encryption key:

$$Key[1] = hash(Key[0], d[0]).$$



**Figure 7.** Key generation concept in dynamic key replacement for AES encryption.

This dynamic key replacement ensures that  $Key[0]$  is never reused for encryption, thereby mitigating potential leakage of AES characteristics and reducing SCA risks.

The hash function, implemented in the HashAESKey method, dynamically updates the AES key to ensure enhanced security while maintaining computational efficiency for IoT devices. The process works as follows: The hash function takes  $d[0]$  (the 32-bit hopping operator) as the salt and processes it to modify  $Key[0]$ . The AES key is represented as a 16-byte array, and the hashing process iterates over these 16 bytes in four 4-byte blocks. For each byte, a bitwise XOR operation is performed between the byte and the least significant 8 bits of the salt. After each XOR operation, the salt is right-shifted by 4 bits to ensure all bits contribute to the transformation. This operation produces  $Key[1]$ , which is used for subsequent encryption cycles.

The algorithm can be summarized as follows:

- Step 1: initialize the salt with  $d[0]$ .
- Step 2: iterate over the 16 bytes of  $Key[0]$  in four 4-byte blocks.
- Step 3: for each byte, apply a bitwise XOR with the least significant 8 bits of the salt.
- Step 4: right-shift the salt by 4 bits after processing each byte.
- Step 5: update the corresponding byte in  $Key[0]$  to produce  $Key[1]$ .

This lightweight hashing process ensures that the updated key incorporates sufficient randomness and unpredictability derived from  $d[0]$ , while keeping computational overhead minimal. By leveraging this approach, the dynamic key replacement mechanism prevents key reuse and enhances resilience against SCAs, ensuring secure communication in resource-constrained IoT environments.

- Dynamic Key Replacement Process

The key replacement process, illustrated in Figure 8, is triggered after every  $M$  data transmission. A new hopping operator is generated using the D-H key exchange, which derives a fresh encryption key to prevent key reuse. For enhanced randomness, the private keys in the D-H exchange are generated from high-quality random numbers or, in offline scenarios, through analog value readings from the microcontroller's pins, such as using the Arduino analogRead() function.

```

[11-27 12:45:22] Message number      : 2099
[11-27 12:45:22] Incoming message   : 993B7BC6B34C9DFA48E82502132063A8
[11-27 12:45:22] Current AES key    : 65 CD F3 C2 10 C6 E0 D6 1C F3 83 AF 0C C5 D1 8C
[11-27 12:45:22] Plain text        : Hello world
[11-27 12:45:22] Executing D-H...
[11-27 12:45:22] Generate private key : 715652219
[11-27 12:45:22] Generate public key  : 241877510
[11-27 12:45:22] Received public key  : 1632961243
[11-27 12:45:22] Shared key          : 1511346633
[11-27 12:45:22] Message number      : 2100
[11-27 12:45:22] Incoming message   : 96EC9F094DA5706B477931B19832120E
[11-27 12:45:22] Current AES key    : AC D1 A2 97 D9 DA B1 83 D5 EF D2 FA C5 D9 80 D9
[11-27 12:45:22] Plain text        : Hello world
    
```

A new AES key (AC D1 A2 97 D9 DA 81 83 05 EF D2 FA C5 D9 80 D9) is dynamically generated based on the shared key derived from the D-H key exchange.

Figure 8. D-H key exchange.

This mechanism significantly strengthens IoT system security by ensuring that each transmission uses a unique encryption key, effectively thwarting SCAs that exploit predictable electromagnetic signals.

- D-H Key Exchange Implementation

The lightweight implementation of the D-H key exchange is specifically tailored for resource-constrained microcontrollers, such as the Arduino Uno. This implementation utilizes carefully chosen parameters to strike an optimal balance between security and computational efficiency: a generator ( $G = 37$ ) and a prime modulus ( $P = 2,147,483,647$ ).

The rationale behind choosing these parameters is rooted in their ability to meet the security and efficiency requirements of IoT devices. The prime modulus  $P = 2,147,483,647$  is a 31-bit prime number that provides a robust foundation for the discrete logarithm problem, which is the basis of the D-H protocol. While traditional implementations often use larger primes to achieve higher security levels, the 31-bit prime is more suitable for IoT applications where computational efficiency is critical. This size of  $P$  ensures that brute-force attacks remain computationally infeasible, even when attempted on adversarial hardware.

Similarly, the generator  $G = 37$  was selected as a small, fixed integer to optimize the modular exponentiation process. Using a smaller generator reduces the computational burden on microcontrollers, allowing for faster key calculations while maintaining adequate randomness and security. Additionally, this choice ensures a uniform distribution of the generated keys across the finite field defined by the modulus  $P$ , which is essential for preserving cryptographic strength.

The combination of  $G = 37$  and  $P = 2,147,483,647$  provides an effective trade-off between security and performance, making it particularly well-suited for lightweight microcontrollers. In IoT scenarios, the time and energy constraints of adversaries, coupled with the real-time operational demands of IoT devices, further mitigate the risks of brute-force and similar attacks. These parameters are not only computationally efficient but also sufficiently secure for practical deployment in real-world IoT applications, such as access control systems and sensor networks.

To implement this lightweight D-H key exchange, the public keys are computed by each party, using their private keys as  $PK_A = G^a \text{ mod } P$  for the IoT device and  $PK_B = G^b \text{ mod } P$  for the server. Once the public keys are exchanged, the shared symmetric key is derived as  $SK = PK_B^a \text{ mod } P = PK_A^b \text{ mod } P$ . This ensures that both parties compute the same shared key due to the commutative property of modular exponentia-

tion. The detailed step-by-step process of this key exchange is outlined in Table 4, which illustrates the sequence of operations required to establish the shared symmetric key.

**Table 4.** Lightweight D-H key exchange process for IoT devices.

IoT Device (A)	Public Information	Server (B)
Private Key Selection $a = 777$	$G = 37$ $P = 2, 147, 483, 647$	Private Key Selection $b = 888$
Public Key Calculation $PK_A = G^a \text{ mod } P$ $= 37^{777} \text{ mod } 2, 147, 483, 647$ $= "1, 348, 037, 377"$		Public Key Calculation $PK_B = G^b \text{ mod } P$ $= 37^{888} \text{ mod } 2, 147, 483, 647$ $= "37, 387, 895"$
Public Key Exchange	$PK_A = "1, 348, 037, 377"$ $PK_B = "37, 387, 895"$	Public Key Exchange
Shared Secret Calculation $SK = PK_B^a \text{ mod } P$ $= 37, 387, 895^{777} \text{ mod } 2, 147, 483, 647$ $= "1, 142, 936, 476"$		Shared Secret Calculation $SK = PK_A^b \text{ mod } P$ $= 1, 348, 037, 377^{888} \text{ mod } 2, 147, 483, 647$ $= "1, 142, 936, 476"$

This lightweight implementation of the D-H key exchange ensures secure key distribution, even for devices with limited processing power, by balancing computational efficiency and cryptographic strength. Through the careful selection of parameters and optimization of the microcontroller’s workload, this approach provides a highly efficient solution for IoT environments, where resource constraints are a critical consideration. It enables secure cryptographic key exchanges without compromising the performance of resource-constrained devices, such as the Arduino Uno.

The proposed method not only meets the security demands of modern IoT applications but also demonstrates excellent adaptability to the constraints of real-world deployments, where both energy consumption and processing capabilities are limited. This lightweight D-H mechanism is ideal for scalable IoT networks that require robust security without sacrificing performance.

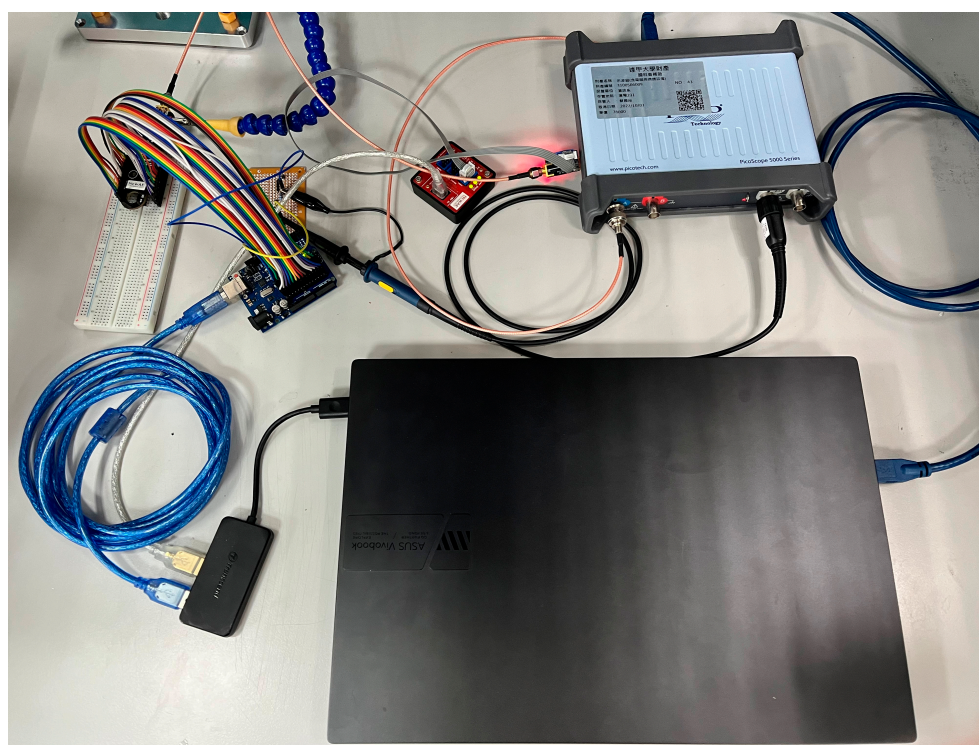
### 5. Experimental Results

This chapter presents a detailed analysis of the experimental results for the proposed “Efficient Key place Mechanism for Lightweight IoT Microcontrollers”. The experiment aimed to verify the correctness of the AES encryption and decryption processes, evaluate the time efficiency of the dynamic key replacement mechanism, and assess the effectiveness of different key replacement frequencies in defending against SCAs. Through the analysis of experimental data, we demonstrate the mechanism’s actual performance in enhancing the security and operational efficiency of IoT devices. The following sections detail the experimental design and results.

#### 5.1. Experimental Setup

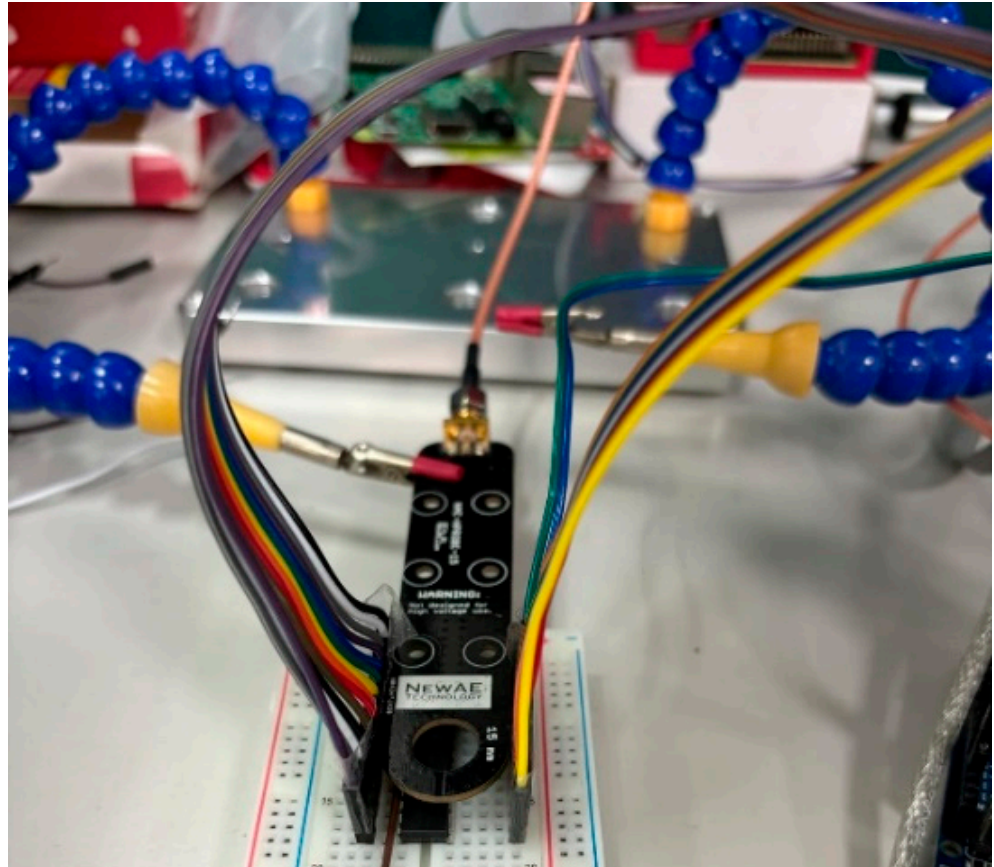
The experimental setup, as shown in Figure 9a, establishes a testing platform for lightweight IoT devices to simulate and evaluate the encryption key protection mechanism and its ability to resist SCAs. In this experiment, an Arduino UNO was used as the client device for data transmission and was set as the target for SCAs. The server side was implemented by Java, responsible for receiving and verifying data from the client to ensure the integrity and correctness of the transmission. The attack platform was configured based on the methods proposed in [31], with both hardware and software set up accordingly. During the attack, the H-Field probe used in this study was the CW505

Planar H-Field Probe, manufactured by NewAE Technology Inc. This probe is specifically designed for side-channel analysis and is capable of capturing electromagnetic radiation emitted by the target device during operations such as AES encryption. The CW505 probe was positioned near the client device to monitor and collect the electromagnetic signals generated during the encryption process, providing critical data for analyzing potential side-channel vulnerabilities. These signals were recorded as trace data for further analysis. Figure 9b provides a physical image of the electromagnetic signals being captured via a loop antenna, which were processed in real time using the PicoScope 5244B oscilloscope. The captured traces were transmitted to the control computer for formatting and preprocessing, and subsequently sent to the computing server for further analysis. The server ran a CPA program, following the configuration by Peng et al. [32], to conduct attack analysis and return results. This setup effectively simulates real-world SCA scenarios in an IoT environment and validates the feasibility and effectiveness of the proposed experimental methodology. Specifically, the CPA uses Pearson's correlation coefficient to determine the linear relationship between the hypothetical power consumption values (e.g., Hamming Weight or Hamming Distance models) and the recorded side-channel traces. The correlation coefficient is calculated using Equation (1). Here  $r_{j,k,t}$  represents the correlation coefficient for the  $j$ -th subkey, the  $k$ th key guess, and the  $t$ -th sample point of the trace.  $x$  refers to the hypothetical power values,  $y$  refers to the recorded side-channel signal, and  $\bar{x}, \bar{y}$  are their respective averages. By analyzing the resulting correlation matrix  $R_j$ , the element with the highest correlation value identifies the correct subkey.



(a) Whole environment

Figure 9. Cont.



(b) Physical image of signal captured by probes

**Figure 9.** Experimental environment setup.

### 5.2. Verification of AES Encryption and Decryption

Before conducting SCAs, we first verified the correctness of data encryption and decryption between the client side and server side. By the lightweight D-H protocol, an AES-encrypted channel was established between the Arduino UNO and the control computer by  $Key[0]$ ; this channel ensured the confidentiality and integrity of transmitted data. After successfully establishing the secure channel, encrypted test messages were transmitted to the server. During the transmission process, Wireshark was employed to monitor and analyze the encrypted message packets. Figures 10 and 11 illustrate the captured network traffic, confirming the successful delivery of encrypted packets from the IoT device to the server. The analysis verified key indicators such as packet size, transmission sequence number, and destination address, ensuring that packets were neither altered nor dropped during transit.

Figure 11 highlights the numbering of each transmitted data packet. For instance, the packet labeled “2100” represents the 2100th AES-128 encrypted message. A comparison of packets 2098, 2099, and 2100 reveals that the encryption key for the 2100th packet was dynamically replaced with a new key: “AC D1 A2 97 DA B1 83 D5 EF D2 FA C5 D9 80 D9”. This validates the functionality of the dynamic AES key replacement mechanism, demonstrating its role in enhancing transmission security by using a unique encryption key for each subsequent packet.



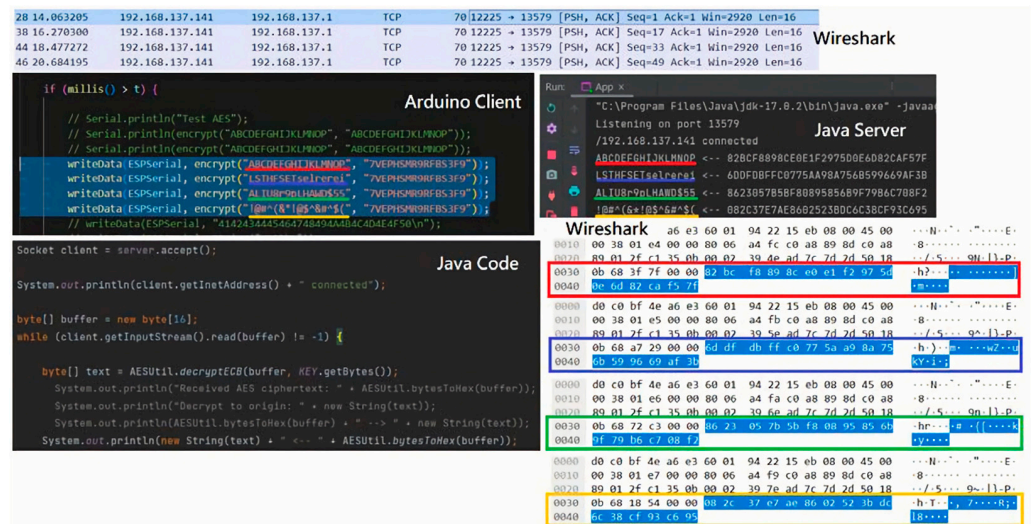


Figure 10. Server-side secure channel setup and encrypted packets transmission.

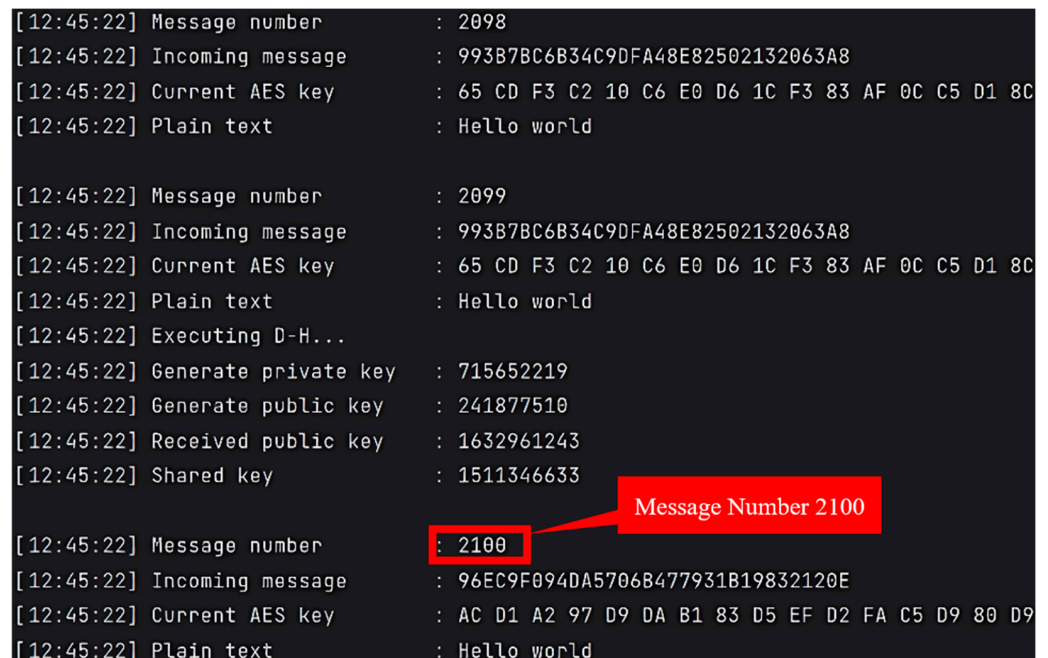


Figure 11. Process of server side receiving encrypted data and decryption.

The figure further confirms that the dynamically replaced key was successfully used by the server to decrypt the encrypted packets. For example, the decrypted content of Message Number 2100 matches the expected plaintext message, verifying the accuracy of the encryption and decryption processes on both the client and server sides. This successful decryption not only confirms the correctness of the encrypted packets but also demonstrates the mechanism’s reliability and integrity throughout the transmission process.

Additionally, Figure 11 demonstrates that the dynamic AES key replacement mechanism maintains the correct sequence and integrity of the transmitted packets. The proper order of packets (e.g., 2098, 2099, and 2100) and their successful decryption validate that no data were lost or corrupted during transit. The server consistently verifies and processes the packets, ensuring reliable and secure communication.

In summary, the analysis presented in Figures 10 and 11 underscores the effectiveness of the proposed dynamic AES key replacement mechanism. The accurate decryption of

Message Number 2100 serves as strong evidence of the mechanism’s capability to ensure data integrity, reliability, and security across the entire communication process.

### 5.3. Time Efficiency Analysis of Dynamic Key Replacement

Given the computational limitations of microcontrollers in IoT devices, evaluating the time efficiency of the proposed key replacement mechanism is essential to ensure it operates effectively in resource-constrained environments. To this end, experiments were conducted on an Arduino UNO development board to assess the performance of the key replacement mechanism at various frequencies. During the tests, plaintext messages were encrypted using AES-128, with the encryption key replaced at different intervals. For each scenario, a total of 3000 encrypted packets were transmitted, and the time required for each transmission was recorded to measure the time overhead associated with different key replacement frequencies.

Table 5 presents a detailed breakdown of the average transmission time per packet when the key was replaced after every 1, 10, 30, and 50 packets. The results indicate that replacing the key for every transmission increases the average time per packet from 12 ms (with infrequent replacements) to 50 ms. Although this represents a measurable increase, it remains well within acceptable limits for real-world applications. To provide context, we compared these results to typical access control systems, where the average time for a card swipe operation ranges between 0.5 and 1 s. Even with the most frequent key replacement scenario, the computational overhead remains negligible compared to the user interaction time in such systems.

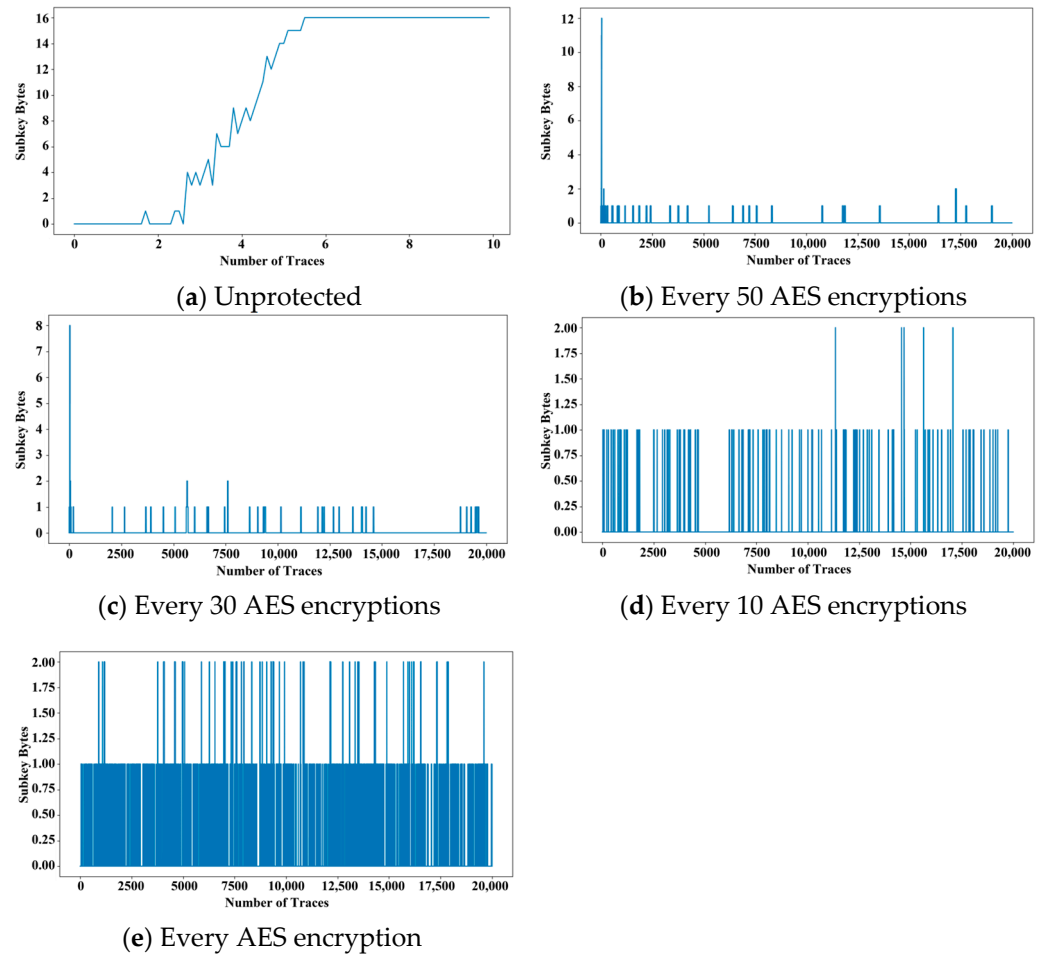
**Table 5.** Time efficiency comparison of different key replacement frequencies.

D-H Key Exchange Frequency	Every 1 Message	Every 10 Messages	Every 30 Messages	Every 50 Messages
Total time for transmitting 3000 encrypted messages (ms)	150,995	69,011	45,415	36,969
Average time per message (ms)	50	23	15	12

The experimental results confirm that the proposed key replacement mechanism effectively enhances security by preventing key reuse while maintaining computational feasibility. The mechanism’s average transmission time across all tested frequencies demonstrates its adaptability to resource-constrained environments. Additionally, the low transmission times validate that frequent key replacements do not degrade IoT device performance, making this mechanism highly suitable for practical applications in smart access control systems and other low-latency IoT environments.

### 5.4. SCA Results with Different Key Replacement Frequencies

In this study, a wireless loop antenna was employed to receive characteristic signals, SCAs were executed on an Arduino UNO microcontroller, and electromagnetic signals were captured under the experimental conditions depicted in Figure 9. Correlation analysis was utilized to assess the efficacy of the attack in compromising the cryptographic keys. Figure 12a presents the results of an SCA executed on the microcontroller during AES encryption, without any protective countermeasures in place. The x-axis represents the number of data traces required to carry out the attack, while the y-axis indicates the number of successfully recovered key bytes. Our findings reveal that, without protection, an attacker can fully recover all 16 subkeys of the AES-128 encryption algorithm with just approximately 55 traces.



**Figure 12.** SCA results of AES encryption with different key replacements.

To further evaluate the efficacy of the proposed dynamic key replacement mechanism, we executed SCAs at various intervals of encryption operations, modifying the encryption key at different frequencies. The results as shown in Figure 12b–e illustrate a significant reduction in the effectiveness of the SCAs as the frequency of key replacement increases. Table 6 summarizes these findings, underscoring the impact that key replacement frequency has on an attacker’s ability to compromise the encryption keys.

**Table 6.** Results of SCAs with different key replacement frequencies.

Key Replacement Frequency	Without Key Replacement	1	10	30	50
Number of traces required	55	>20,000	>20,000	>20,000	>20,000
Number of subkeys compromised	16	2	2	8	12

- Figure 12a: without any protection.
- Figure 12b: when the key was replaced every 50 encryption operations, the attacker was able to successfully retrieve 12 subkeys.
- Figure 12c: at a replacement interval of 30 encryption operations, only eight subkeys were compromised.
- Figure 12d: reducing the key replacement interval to 10 encryption operations resulted in only two compromised subkeys.
- Figure 12e: Replacing the key after every single encryption cycle limited the attacker’s success to just two subkeys.

These results clearly demonstrate the inverse relationship between key replacement frequency and the success rate of SCAs. The more frequently the encryption key is replaced, the more difficult it becomes for the attacker to successfully compromise the cryptographic keys. Notably, the strategies of replacing the key after every one and ten encryption operations proved particularly effective in thwarting template attacks, which are often capable of breaching encryption with minimal traces.

As reported by Wu et al. [33], deep learning and template attack techniques can allow attackers to recover encryption keys using only a small number of traces. However, the high-frequency key replacement mechanism proposed in this study has substantially reduced the number of subkeys that could be successfully retrieved by such attacks. The results provide compelling evidence that the dynamic key replacement mechanism significantly enhances the security of IoT devices, bolstering their resistance to SCAs and ensuring robust protection of sensitive data.

### 5.5. Scalability of the Proposed Mechanism in IoT Environments

The proposed mechanism demonstrates strong scalability, as each IoT device independently derives encryption keys using the lightweight D-H protocol. In high-density IoT environments, the computational overhead on microcontrollers remains minimal, ensuring the feasibility of large-scale deployments.

To further validate the mechanism's scalability, additional experiments were conducted by simulating the connection of five additional devices in a test environment. Each device established a secure communication channel with the server using the proposed mechanism. The results showed that the average latency per device increased by only 3 milliseconds during key negotiation, even with the added device density. This negligible increase in latency highlights the mechanism's practicality for real-world implementations, ensuring secure and efficient communication as the number of connected devices grows.

### 5.6. Security Analysis

The proposed lightweight Diffie–Hellman (D-H) protocol is specifically designed to address the unique security challenges of resource-constrained IoT environments. This section presents a comprehensive analysis of the protocol's security properties, with a focus on ensuring confidentiality, authentication, integrity, and resistance to known attacks.

#### 5.6.1. Cryptographic Strength of Parameters

As detailed in Section 4.4, the proposed protocol employs carefully selected parameters: the prime modulus  $p = 2,147,483,647$  and the generator  $G = 37$ . These choices achieve an optimal balance between computational efficiency and cryptographic strength, making the protocol well-suited for resource-constrained IoT devices. The selected parameters provide a robust cryptographic foundation, ensuring security without compromising performance.

#### 5.6.2. Confidentiality

The protocol guarantees the confidentiality of shared symmetric keys derived through the D-H key exchange mechanism. Even if attackers intercept the public keys ( $PK_A$  and  $PK_B$ ), the difficulty of solving the discrete logarithm problem (DLP) ensures that the private keys ( $a$  and  $b$ ) and the derived shared key remain secure. This ensures the confidentiality of key exchange communications, protecting them from unauthorized access.

#### 5.6.3. Authentication

Although the protocol does not inherently provide an explicit authentication mechanism, it assumes either a secure initial key exchange or the involvement of a trusted third party for public key verification. Without such measures, the protocol remains vulnerable

to man-in-the-middle (MITM) attacks. To mitigate this risk, integrating a digital signature or certificate-based authentication mechanism is recommended. This would ensure the authenticity of public keys ( $PK_A$  and  $PK_B$ ), preventing impersonation by malicious actors and safeguarding the key exchange process.

#### 5.6.4. Integrity

The protocol maintains the integrity of the derived shared key through consistent modular exponentiation. However, to safeguard the integrity of transmitted data, it is recommended to incorporate a MAC derived from the shared key ( $SK$ ). This additional layer of protection ensures that any tampering with encrypted messages can be detected and mitigated, enhancing overall communication security.

#### 5.6.5. Resistance to Known Attacks

- MITM attacks: Without an explicit authentication mechanism, the protocol may be vulnerable to MITM attacks. To mitigate this risk, enhancements such as pre-shared keys or certificate-based validation can be incorporated, ensuring that attackers cannot intercept or alter public keys during transmission. These measures strengthen the protocol's ability to verify the authenticity of communicating parties.
- Replay attacks: Incorporating time-sensitive parameters or nonces into the key exchange process effectively prevents replay attacks, where attackers attempt to reuse old keys to impersonate legitimate parties. These additions enhance the protocol's robustness in real-world deployments by ensuring that each key exchange session remains unique and resistant to duplication.
- SCAs: While the protocol's mathematical foundation ensures robust cryptographic security, its implementation on resource-constrained IoT devices may leave it vulnerable to SCAs, such as power analysis or electromagnetic leakage. To address this, the proposed dynamic key replacement mechanism mitigates the risk of SCAs by dynamically updating encryption keys after each cryptographic operation. This process ensures that even if partial information is leaked through physical signals, it becomes obsolete before it can be exploited. As a result, the mechanism significantly enhances the protocol's resilience against SCAs, providing stronger protection for IoT devices in practical deployments.

#### 5.6.6. Scalability and Real-World Deployment

The D-H protocol demonstrates excellent scalability in IoT environments. Each device independently computes its session key using efficient modular exponentiation, minimizing computational overhead. As presented in Section 5.5, experimental results show that even in high-density IoT networks, the average communication delay per device increases by only 3 milliseconds. This minimal overhead underscores the protocol's practicality and suitability for large-scale deployments, ensuring both efficiency and performance in resource-constrained settings.

#### 5.6.7. Comparative Analysis

Compared to traditional D-H implementations that rely on larger prime numbers for enhanced security, the proposed mechanism achieves an optimal balance between security and performance. As highlighted in Table 3, the proposed approach effectively mitigates SCAs while significantly reducing computational complexity. This dual capability makes it particularly well-suited for resource-constrained environments. By addressing key challenges identified in prior research, the proposed mechanism not only strengthens security against SCAs but also ensures practical applicability in IoT deployments, where efficiency and resource optimization are essential.

## 6. Conclusions

In the IoT scenario, encryption is important for keeping data safe and defending against attacks. If SCAs are not stopped, encryption keys can be stolen, which means sensitive information can be accessed and the whole system could be at risk. Traditional hardware-level defense mechanisms are not suitable for IoT devices because of the unaffordable cost like power overhead or circuit area overhead. Consequently, it is imperative to implement software-based defense strategies to guarantee system security without impairing computational performance.

In this study, we proposed a dynamic key replacement mechanism for AES encryption, which effectively enhances the system's resilience against SCAs by modifying the encryption keys before an attack can successfully recover them. A lightweight key replacement mechanism was designed based on D-H key exchange protocol, which is straightforward to implement and particularly well-suited for resource-constrained microcontroller environments. This mechanism is not limited to AES encryption but can also be applied to other encryption systems requiring key negotiation.

The experimental results demonstrate that as the frequency of key replacement increases, the system's ability to withstand SCAs is significantly enhanced. Specifically, when the key was replaced after every one or ten encryptions, the success rate of SCA dropped substantially. These findings confirm the effectiveness of the dynamic key replacement mechanism in enhancing the security of IoT devices against SCAs.

**Author Contributions:** Conceptualization, C.-W.K.; methodology, C.-W.K. and W.W.; software, W.W.; validation, C.-C.L., Y.-Y.H. and J.-R.L.; formal analysis, W.W. and K.-Y.T.; investigation, W.W. and C.-C.L.; resources, C.-W.K.; data curation, C.-W.K. and W.W.; writing—original draft preparation, C.-W.K. and W.W.; writing—review and editing, C.-W.K., W.W. and C.-C.L.; visualization, C.-C.L.; supervision, K.-Y.T.; project administration, C.-W.K.; funding acquisition, C.-W.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research work funded by National Science and Technology Council, Taiwan, under Grant NSTC 113-2221-E-035-075.

**Data Availability Statement:** Dataset available on request from the author.

**Acknowledgments:** This work was supported in part by the Longmau Technology Co., Ltd., under Grant 12B1001T.

**Conflicts of Interest:** We declare that we have no conflict of interest to report regarding the present study.

## References

1. Global Cellular IoT Connections Surpassed 4 Billion in 2024, Driven by 5G and LTE Cat 1 Bis. Available online: <https://iot-analytics.com/global-cellular-iot-connections/> (accessed on 27 December 2024).
2. Moini, S.; Tian, S.; Holcomb, D.; Szefer, J.; Tessier, R. Power Side-Channel Attacks on BNN Accelerators in Remote FPGAs. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2021**, *11*, 357–370. [CrossRef]
3. Yilmaz, B.B.; Prvulovic, M.; Zajić, A. Electromagnetic Side Channel Information Leakage Created by Execution of Series of Instructions in a Computer Processor. *IEEE Trans. Inf. Forensics Secur.* **2019**, *15*, 776–789. [CrossRef]
4. Aljuffri, A.; Zwalua, M.; Reinbrecht, C.R.W.; Hamdioui, S.; Taouil, M. Applying Thermal Side-Channel Attacks on Asymmetric Cryptography. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, *29*, 1930–1942. [CrossRef]
5. Schaumont, P.; Tiri, K. Masking and Dual-Rail Logic Don't Add Up. In Proceedings of the Cryptographic Hardware and Embedded Systems (CHES 2007), Vienna, Austria, 10–13 September 2007.
6. Schramm, K.; Paar, C. Higher Order Masking of the AES. In Proceedings of the Topics in Cryptology-CT-RSA 2006, San José, CA, USA, 13–17 February 2006.

7. Baseri, Y.; Chouhan, V.; Ghorbani, A. Cybersecurity in the Quantum Era: Assessing the Impact of Quantum Computing on Infrastructure. *arXiv* **2024**, arXiv:2404.10659.
8. Chng, S.; Lu, H.Y.; Kumar, A.; Yau, D. Hacker types, motivations and strategies: A comprehensive framework. *Comput. Hum. Behav. Rep.* **2022**, *5*, 100167. [[CrossRef](#)]
9. Munoz, P.S.; Tran, N.; Craig, B.; Dezfouli, B.; Liu, Y. Analyzing the Resource Utilization of AES Encryption on IoT Devices. In Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC 2018), Honolulu, HI, USA, 12–15 November 2018.
10. Fatima, S.; Rehman, T.; Fatima, M.; Khan, S.; Ali, M.A. Comparative Analysis of Aes and Rsa Algorithms for Data Security in Cloud Computing, In Proceedings of the 7th International Electrical Engineering Conference (IEEC 2022). Karachi, Pakistan, 25–26 March 2022.
11. Devi, M.; Majumder, A. *Side-Channel Attack in Internet of Things: A Survey*, In *Applications of Internet of Things*; Springer: Singapore, 2021; pp. 213–222.
12. Xing, Z.; Zhao, B.; Xu, B.; Ren, G.; Liu, Z. Enhanced Message Authentication Encryption Scheme Based on Physical-Layer Key Generation in Resource-Limited Internet of Things. *KSII Trans. Internet Inf. Syst. (TIIS)* **2024**, *18*, 2546–2563.
13. Rawat, J.; Kumar, I.; Mohd, N.; Rana, K.K.S.; Pathak, N.; Gupta, R.K. IoT-Based Home Automation System Using ESP8266. In Proceedings of the International Conference on Innovative Computing and Communications (ICICC 2023), New Delhi, India, 17–18 February 2023.
14. Vuppala, S.; Mady, A.E.; Kuenzi, A. Moving Target Defense Mechanism for Side-Channel Attacks. *IEEE Syst. J.* **2020**, *14*, 1810–1819. [[CrossRef](#)]
15. Daemen, J.; Rijmen, V. *The Design of Rijndael*, 2nd ed.; Springer: Singapore, 2020.
16. Kuo, C.W.; Tsai, K.Y.; Weng, W.M.; Lin, C.C.; Hong, Y.Y.; Wang, G.L. Implementation and Analysis of Side-Channel Attack Mitigation Based on Autoencoder. *Commun. CCISA* **2023**, *29*, 1–18.
17. Lightweight Cryptography Standardization Process: NIST Selects Ascon. Available online: <https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon> (accessed on 23 July 2023).
18. Kocher, P.; Jaffe, J.; Jun, B. Differential Power Analysis. In Proceedings of the Advances in Cryptology—CRYPTO’99, Santa Barbara, CA, USA, 15–19 August 1999.
19. Brier, E.; Clavier, C.; Olivier, F. Correlation Power Analysis with a Leakage Model. In Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2004, Cambridge, MA, USA, 11–13 August 2004.
20. Messerges, T.S. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2000, Worcester, MA, USA, 17–18 August 2000.
21. Zhang, Q.; Wang, A.; Niu, Y.; Shang, N.; Xu, R.; Zhang, G.; Zhu, L. Side-Channel Attacks and Countermeasures for Identity-Based Cryptographic Algorithm SM9. *Secur. Commun. Netw.* **2018**, *2018*, 1–14. [[CrossRef](#)]
22. Diffie, W.; Hellman, M.E. New Directions in Cryptography. *IEEE Trans. Inf. Theory* **1976**, *22*, 644–654. [[CrossRef](#)]
23. Santoso, F.K.; Vun, N.C.H. Securing IoT for smart home system. In Proceedings of the International Symposium on Consumer Electronics (ISCE 2015), Madrid, Spain, 24–26 June 2015.
24. Collotta, M.; Pau, G. A Novel Energy Management Approach for Smart Homes Using Bluetooth Low Energy. *IEEE J. Sel. Areas Commun.* **2015**, *33*, 2988–2996. [[CrossRef](#)]
25. Zualkernan, I.A.; Al-Ali, A.R.; Jabbar, M.A.; Zabalawi, I.; Wasfy, A. InfoPods: Zigbee-Based Remote Information Monitoring Devices for Smart-Homes. *IEEE Trans. Consum. Electron.* **2009**, *55*, 1221–1226. [[CrossRef](#)]
26. Tsai, K.-L.; Leu, F.-Y.; You, I.; Chang, S.-W.; Hu, S.-J.; Park, H. Low-Power AES Data Encryption Architecture for a LoRaWAN. *IEEE Access* **2019**, *7*, 146348–146357. [[CrossRef](#)]
27. Mangard, S.; Oswald, E.; Popp, T. *Power Analysis Attacks*, 1st ed.; Springer: New York, NY, USA, 2007.
28. Nagata, M.; Miki, T.; Miura, N. Physical attack protection techniques for IC chip level hardware security. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2022**, *30*, 5–14. [[CrossRef](#)]
29. Bogdanov, A.; Knudsen, L.R.; Leander, G.; Paar, C.; Poschmann, A.; Robshaw, M.J.B.; Seurin, Y.; Vikkelsoe, C. PRESENT: An ultra-lightweight block cipher. In Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2007: 9th International Workshop, Vienna, Austria, 10–13 September 2007.
30. Tsai, K.L.; Huang, Y.L.; Leu, F.Y.; You, I.; Huang, Y.L.; Tsai, C.H. AES-128 based secure low power communication for LoRaWAN IoT environments. *IEEE Access* **2018**, *6*, 45325–45334. [[CrossRef](#)]
31. Kuo, C.W.; Lin, C.C.; Hong, Y.Y.; Liu, J.R.; Yeh, C.H.; Tsai, K.Y. Research and Analysis of the Effects of Different Shielding Materials on Resisting Side-Channel Attacks on IoT Device Microcontroller. In Proceedings of the 8th International Conference on Cryptography, Security and Privacy (CSP 2024), Osaka, Japan, 20–22 April 2024.

32. Peng, S.Y.; Hong, W.C.; Li, J.T.; Huang, S.J. Framework for efficient SCA resistance verification of IoT devices. In Proceedings of the IEEE International Conference on Applied System Invention (ICASI 2018), Chiba, Japan, 13–17 April 2018.
33. Wu, L.; Perin, G.; Picek, S. The best of two worlds: Deep learning-assisted template attack. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**, *3*, 413–437. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.