Fan YH, Wang MQ, Li YB *et al.* A secure IoT firmware update scheme against SCPA and DoS attacks. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 36(2): 419–433 Mar. 2021. DOI 10.1007/s11390-020-9831-8

A Secure IoT Firmware Update Scheme Against SCPA and DoS Attacks

Yan-Hong Fan, Mei-Qin Wang*, Yan-Bin Li, Kai Hu, and Mu-Zhou Li

School of Cyber Science and Technology, Shandong University, Qingdao 266237, China

Key Laboratory of Cryptologic Technology and Information Security (Shandong University), Ministry of Education Qingdao 266237, China

E-mail: fanyh@mail.sdu.edu.cn; mqwang@sdu.edu.cn; liyanbin1015@126.com; {hukai, muzhouli}@mail.sdu.edu.cn

Received July 9, 2019; accepted February 24, 2020.

Abstract In the IEEE S&P 2017, Ronen *et al.* exploited side-channel power analysis (SCPA) and approximately 5 000 power traces to recover the global AES-CCM key that Philip Hue lamps use to decrypt and authenticate new firmware. Based on the recovered key, the attacker could create a malicious firmware update and load it to Philip Hue lamps to cause Internet of Things (IoT) security issues. Inspired by the work of Ronen *et al.*, we propose an AES-CCM-based firmware update scheme against SCPA and denial of service (DoS) attacks. The proposed scheme applied in IoT terminal devices includes two aspects of design (i.e., bootloader and application layer). Firstly, in the bootloader, the number of updates per unit time is limited to prevent the attacker from acquiring a sufficient number of useful traces in a short time, which can effectively counter an SCPA attack. Secondly, in the application layer, using the proposed handshake protocol, the IoT device can access the IoT server to regain update permission, which can defend against DoS attacks. Moreover, on the STM32F405+M25P40 hardware platform, we implement Philips' and the proposed modified schemes. Experimental results show that compared with the firmware update scheme of Philips Hue smart lamps, the proposed scheme additionally requires only 2.35 KB of Flash memory and a maximum of 0.32 s update time to effectively enhance the security of the AES-CCM-based firmware update process.

Keywords Internet of Things, firmware update, authenticated encryption, side-channel power analysis, denial of service

1 Introduction

In recent years, the number of Internet of Things (IoT) devices introduced in the market has increased drastically. This trend is expected to continue and business analysts predicted that the number of IoT devices in the market will exceed 250 billion units by 2025 [1].

In the IoT world, millions of interconnected smart devices collect and exchange potentially sensitive data. If a new firmware vulnerability is discovered or disclosed in the public community, there are possibilities for malicious attackers to exploit the vulnerability and infect billions of connected devices [2, 3]. Firmware update is a particularly important countermeasure to correct firmware vulnerability remotely and enhance the security of IoT devices [4, 5]. However, the US Federal Trade Commission (FTC) has indicated that there are numerous security risks (e.g., firmware exposure, malicious firmware modification) in the firmware update process①. Malicious attackers could exploit such a vulnerability to control an infected target device remotely, which could then be used to launch a denial of service (DoS) attack or even to violate the privacy of the target device owner [6].

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61572293, 61502276 and 61692276, the National Cryptography Development Foundation of China under Grant No. MMJJ20170102, the Major Scientific and Technological Innovation Projects of Shandong Province of China under Grant No. 2017CXGC0704, and the Natural Science Foundation of Shandong Province of China under Grant No. ZR2016FM22.

*Corresponding Author

①https://www.ftc.gov/news-events/press-releases/2015/01/ftc-report-internet-things-urges-companies-adopt-best-practices, May 2019.

In recent years, there have been attacks on the firmware update process. The IoActive Security Company hacked a counterfeit money detector that had no encryption operations protecting the firmware update process②. Using this vulnerability, the attacker loaded malicious firmware into the detector, causing it to accept counterfeit money. Cui *et al.*[7] presented a firmware modification vulnerability of the HP-RFU LaserJet printer and injected malware into the printer's firmware via standard printed documents. Ronen *et al.*[8] performed a worm infection attack on Philips Hue smart lamps. To facilitate such an attack, they discovered a major bug in the implementation of the Zigbee Light Link protocol and extracted the Advanced Encryption Standard (AES)—Counter with Cipher Block Chaining-Message Authentication Code (CCM) key that Philips Hue smart lamps use to encrypt and authenticate new firmware in the bootloader.

There have been countermeasures proposed to secure firmware updates from different perspectives. For the security of the protocol layer, methods have been suggested in [9–11] to ensure the security of the firmware transmission. In [12], Choi *et al.* introduced a secure firmware validation and update scheme that utilizes an ID-based mutual authentication and key derivation to securely distribute new firmware. In recent years, blockchain has been used in IoT devices to provide secure verification in the firmware release process, for example [13] and [14]. Asokan *et al.*[15] presented an architecture to secure the firmware update of realistic IoT devices that provides end-to-end security between the manufacturers and devices. Further, other authenticated encryption (AE) schemes or block ciphers have been introduced into firmware update encryption mechanisms in IoT devices to prevent firmware exposure and malicious firmware modification.

However, even protected by industry-standard cryptographic techniques, the firmware update process of IoT terminal devices can be misused by hackers to create a new attack such as a side-channel power analysis (SCPA) attack[8, 16]. In [8], an SCPA attack against AES-CCM used to encrypt and verify firmware updates allowed attackers to encrypt, sign, and upload malicious over the air (OTA) updates to infect the lamps. In the SCPA attack, the attacker uses only cheap and easily obtained equipment costing a few hundred dollars to deduce all the secret cryptographic elements used in the bootloader of Philips Hue smart lamps within a few days. Once the secret elements are obtained, the attackers could create any malicious firmware and upload it into any Philips Hue smart lamp. The SCPA attack requires very low cost and can be easily executed by even a single malicious attacker. Therefore, the SCPA attack is a serious threat to firmware update.

To prevent the entire ecosystem from damage by malicious firmware modification, Ronen *et al.*[8] provided two countermeasures: using unique keys per IoT terminal device or using asymmetric cryptography for the firmware verification. Although these two countermeasures could improve the security of the IoT, they require high cost of implementation and are difficult to implement in practice for the resource-constrained and vast number of IoT terminal devices. In [17], Guillen *et al.* proposed an approach that uses anomaly detection to recognize side-channel attacks. Once an attack is detected, three different measures, 1) modifying the key material, 2) delaying the next response, and 3) disabling the cryptographic core, are executed. However, these measures lead to a DoS attack.

According to the above, the firmware update scheme with low cost of implementation and security against SCPA and DoS attack is worth studying. In this paper, to enhance the security of the firmware update, we propose an AES-CCM-based firmware update scheme that can effectively defend against SCPA and DoS attacks and requires low cost of implementation. The proposed firmware update scheme has reasonable engineering tradeoffs in terms of security and cost of implementation; therefore it can be easily applied into resource-constrained IoT terminal devices.

Our Contributions. We propose a new firmware update scheme against SCPA and DoS attacks. In the proposed scheme, using the SCPA method, the attacker will need several years in acquiring a sufficient number of useful traces to deduce all the secret cryptographic elements (i.e., the key and nonce value) used in decrypting and authenticating new firmware. High time cost makes the SCPA method to become meaningless. Moreover, in order to prevent DoS attacks caused by malicious attackers, this paper proposes a new handshake protocol to ensure that IoT terminal devices can access the IoT server securely to regain service privileges of firmware update.

The proposed scheme includes two aspects of design (i.e., the bootloader and the application) in IoT terminal devices. The contributions of this paper are listed below.

②https://ioactive.com/hacking-a-counterfeit-money-detector-for-fun-and-non-profit/, May 2019.

• *New Cryptographic Design Against SCPA Attack in the Bootloader.* In IoT terminal devices, the main function of the bootloader is to perform firmware updates, which move the firmware data from the off-chip Flash to on-chip Flash. In the firmware update process, the bootloader performs an AE scheme to provide the confidentiality and authenticity of the firmware data. The attacker can target the verification phase of the AE scheme to recover the secret cryptographic elements used in IoT terminal devices by SCPA attacks. Ronen *et al.* used SCPA attacks to successfully recover the key and nonce by acquiring approximately 5 000 power traces in [8]. Note that in the process of recovering the secret cryptographic elements, the corresponding associated data in the firmware of each power trace was required to be the same, and the time required to acquire approximately 5 000 power traces was less than one hour. In [8] the attacker could use very low time cost to break the popular IoT device (i.e., Hpilips Hue smart lamp) by recovering the secret cryptographic elements, which poses a significant threat to the security of firmware update for IoT devices. In order to counter the SCPA attack in [8], the proposed scheme introduces additional operations in the bootloader for validating and storing hash values about the associated data of the firmware to limit the times of acquiring useful power traces by the attacker per unit time. In the proposed scheme, the attacker can obtain no more than five useful traces every 24 hours. Acquiring approximately 5 000 useful power traces only needs no more than one hour in [8], but it would require about three years in the proposed scheme. This overly long time cost prevents an attacker from using the SCPA method to recover the secret cryptographic elements in the firmware update loading process.

• *Access Design Against DoS Attack in the Application Layer.* DoS attacks could exist in previous and the proposed schemes without additional countermeasures. In the proposed schemes, if the table storing the hash values is full, the firmware update operation is not performed, which leads to a DoS attack. To prevent the DoS attack, the proposed design introduces access operations and a new handshake protocol in the application layer. The application layer accesses the table storing the hash values of the associated data every 24 hours. If the table is full, the application layer accesses the IoT cloud server through the proposed new handshake protocol. When the handshake is successful, the application can clear the table to enable the bootloader to perform a firmware update again, which can effectively defend against DoS attacks.

To the best of our knowledge, the proposed design is the first approach that protects the AES-CCM-based firmware update process against SCPA and DoS attacks simultaneously.

Outline. The remainder of this paper is organized as follows. In Section 2, we describe the AE CCM scheme, the firmware update process of IoT terminal devices, and other work related to the breaking of the Philips Hue smart lamp scheme in [8]. In Section 3, we present a new IoT firmware update scheme against SCPA and DoS attacks. The experimental results for implementation cost and update rate are provided in Section 4. Finally, we conclude this paper in Section 5.

2 Preliminaries

The definitions of notations used in this paper are presented in Table 1.

2.1 AE CCM Scheme

CCM is an AE scheme that can protect both the confidentiality and authenticity of data [18]. CCM has been standardized in IPsec③, TLS④, Low Power Bluetooth 4.0⑤, and IEEE 802.15.4 [19]. Owing to the excellent performance of CCM and widespread deployment of AES, AES-CCM is especially suitable for resource-constrained IoT terminal devices such as the popular Philips Hue smart lamps.

CCM consists of two important auxiliary primitives: the counter (CTR) mode and the cipher block chaining (CBC) mode. Before describing CCM itself, we first define these two auxiliary primitives.

The first important auxiliary primitive is the CTR mode. The CTR mode applies a predefined cipher $E_K()$ to a set of input blocks generated by a counter value X to produce m output blocks [20]. The inputs of the CTR mode are X, m, and $E_K()$. Its output is a string S with m blocks. Each block of S is computed by $S_j = E_K(inc_x^j(X))$, where inc_x represents counter incrementation, adding one modulo 2^x to X, with the convention that inc_x^j represents j successive implementations. CTR mode is defined in Algorithm 1.

③https://tools.ietf.org/pdf/rfc4309.pdf, Dec. 2005.

④https://www.rfc-editor.org/pdfrfc/rfc6655.txt.pdf, Jul. 2012.

⑤https://www.bluetooth.com/zh-cn/bluetooth-resources, Apr. 2009.

Table 1. Notations and Definitions

Notation	Definition
A	In the AE CCM scheme, A stands for associated data; in the firmware image, A stands for the protocol-level data which includes firmware version, firmware code length and other configuration parameters
P	In the AE CCM scheme, P stands for plaintext; in the firmware image, P stands for firmware source code
C	In the AE CCM scheme, C stands for ciphertext; in the firmware image, C stands for encrypted firmware code
a	Block length of associated data A, $a \in \mathbb{N}_+$
m	Block length of plaintext P, $m \in \mathbb{N}_+$
A_i	Associated data of block i, $i \in \{1, \ldots, a\}$
P_i	Plaintext of block i, $i \in \{1, \ldots, m\}$
C_i	Ciphertext of block i, $i \in \{1, \ldots, m\}$
N	Nonce value of the AE scheme
K	Key value of the AE scheme
X	A counter value in the CTR mode
l	Bit length of N, $l \in \mathbb{N}_+$
n	Bit length of block size, $n \in \mathbb{N}_+$
T	Tag that is generated in CCM processes
CT	Encrypted value of T
$\perp$	Error symbol that represents verification failure of CCM
IV_0	Initialization vector of CBC
Ctr_i	Counter value of block i
$X\|\|Y$	Concatenation of two bit strings X and Y
$X \oplus Y$	Bitwise exclusive-OR of two bit strings X and Y with the same length
$E_K(M)$	Ciphertext of a cipher under key K applied to plaintext block M
$C(N, i)$	Counter generation function which can format the counter index i and N into a complete data block, $i \in \{1, \ldots, m\}$
$SB()$	SubBytes operation function in AES round function
$SR()$	ShiftRows operation function in AES round function
$MC()$	MixColumns operation function in AES round function
sb_i	$sb_i = SB(S_{tmp})$; sb_i represents 16 bytes of state after applying SubBytes operation on S_{tmp} in round function of block cipher AES, $i \in \{1, 2, 3\}$
H_{ad}	Hash values of A
$FC[\,]$	Counter of failed firmware verifications

Algorithm 1. CTR

Input: $X \in \{0,1\}^n$, $m \in \mathbb{N}_+$, $E_K : \{0,1\}^n \to \{0,1\}^n$
Output: $S \in \{0,1\}^{mn}$

$I \leftarrow X$;
for $1 \leqslant j \leqslant m$ **do**
  $S_j \leftarrow E_K(I)$;
  $I \leftarrow inc_x(I)$;
$S \leftarrow S_1||S_2||\cdots||S_m$;
return S;

The second important auxiliary primitive is the CBC mode. The CBC mode is a confidentiality mode whose encryption process features the combination of a plaintext block with the previous ciphertext block [20]. The CBC mode requires an initialization vector to combine with the first plaintext block. The inputs of the function $CBC(\boldsymbol{IV}_0, P, m)$ are the initialization vector $\boldsymbol{IV}_0$, plaintext P, block length m, and $E_K()$; the output is a string S. The CBC mode is defined in Algorithm 2.

Algorithm 2. CBC

Input: $\boldsymbol{IV}_0 \in \{0,1\}^n$, $P \in \{0,1\}^{mn}$, $m \in \mathbb{N}_+$, $E_K : \{0,1\}^n \to \{0,1\}^n$
Output: $S \in \{0,1\}^n$

$C_0 \leftarrow E_k(P_1 \oplus \boldsymbol{IV}_0)$;
for $2 \leqslant j \leqslant m$ **do**
  $C_j \leftarrow E_K(P_j \oplus C_{j-1})$;
$S \leftarrow C_m$;
return S;

As an AE scheme, CCM consists of two modes: encryption mode and decryption mode. The CCM encryption mode is represented by $CCM.Enc(K, N, A, P)$, which is defined in Algorithm 3. The inputs of CCM encryption are key K, nonce N, associated data A, and plaintext P. Its outputs are ciphertext C and the encrypted value of tag CT.

Algorithm 3. CCM Encryption

Input: $K \in \{0,1\}^n$, $N \in \{0,1\}^l$, $A \in \{0,1\}^{an}$, $P \in \{0,1\}^{mn}$
Output: $T \in \{0,1\}^n, C \in \{0,1\}^{mn}$

$B_0||B_1||\cdots||B_{a+m-1} \leftarrow A||P$;
$Y_0 = E_K(B_0)$;
for $1 \leqslant j \leqslant r$ **do**
  $Y_j \leftarrow E_K(P_j \oplus Y_{j-1})$;
$T \leftarrow Y_r$;
for $0 \leqslant j \leqslant m$ **do**
  $Ctr_j \leftarrow C(N, j)$;
  $S_j \leftarrow E_K(Ctr_j)$;
$S \leftarrow S1||S_2||\cdots||S_m$;
$C \leftarrow P \oplus S$;
$CT \leftarrow T \oplus S_0$;
return $C||CT$;

The CCM decryption mode is represented by $CCM.Dec(K, N, A, C, CT)$, which is defined in Algorithm 4. The inputs of CCM decryption are key K, nonce N, associated data A, ciphertext C, and the encrypted value of tag CT. The output is either plaintext P or an error symbol $\perp$.

CCM is displayed in Fig.1. The CCM encryption mode in Fig.1(a) is used to generate encrypted

Algorithm 4. CCM Decryption

Input: $K \in \{0,1\}^n$, $N \in \{0,1\}^l$, $A \in \{0,1\}^{an}$, $C \in \{0,1\}^{mn}$, $CT \in \{0,1\}^n$
Output: $P \in \{0,1\}^{mn}$ or $\{\perp\}$

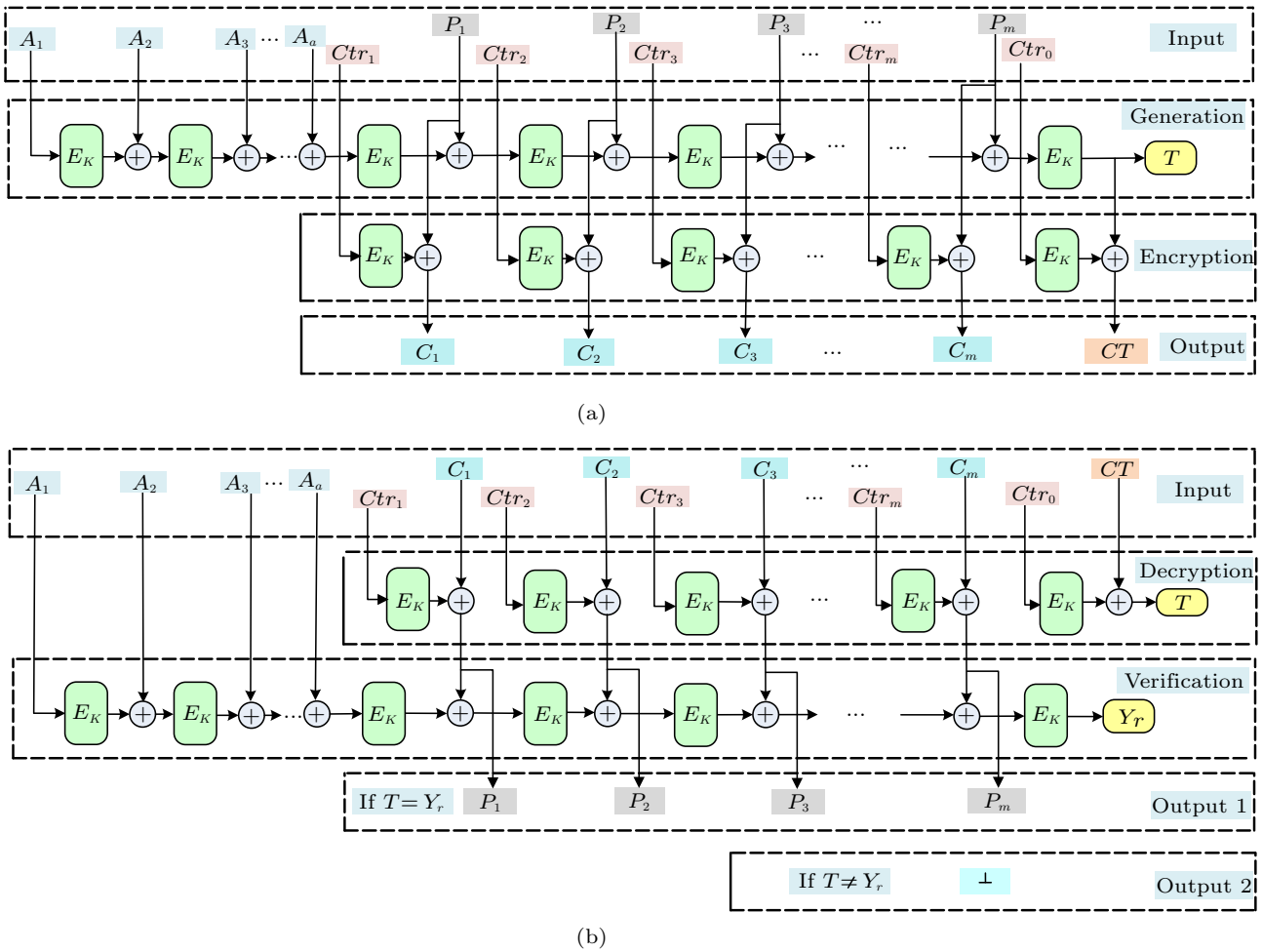
for $0 \leqslant j \leqslant m$ **do**
  $Ctr_j \leftarrow C(N, j)$;
  $S_j \leftarrow E_K(Ctr_j)$;
$S \leftarrow S_1||S_2||\cdots||S_m$;
$P \leftarrow C \oplus S$;
$T \leftarrow CT \oplus S_0$;
$B_0||B_1||\cdots||B_{m+a-1} \leftarrow A||N||P$;
$Y_0 = E_K(B_0)$;
for $1 \leqslant j \leqslant r$ **do**
  $Y_j \leftarrow E_K(P_j \oplus Y_{j-1})$;
if $(Y_r == T)$ **then**
  return P;
else
  return $\perp$;

firmware which includes encrypted firmware code C_i $(i \in \{1, \ldots, m\})$ and encrypted tag value CT. The CCM decryption mode in Fig.1(b) is used to decrypt encrypted firmware and verify the authenticity of firmware source code P_i $(i \in \{1, \ldots, m\})$ and the associated data A_i $(i \in \{1, \ldots, a\})$. If verification succeeds, the CCM decryption mode returns firmware source code P_i $(i \in \{1, \ldots, m\})$. Otherwise, only the error symbol $\perp$ is returned.

2.2 IoT Firmware Update Process

An IoT manufacturer will release a new version of firmware in a timely manner to add a new function or fix a vulnerability. After this new firmware packet is uploaded on the IoT cloud server, the firmware update can be remotely performed. The detail of a typical IoT firmware update process is given in Appendix.

Ronen *et al.*[8] recovered the key and nonce value of AES-CCM in the bootloader of the Philips Hue smart lamps by an SCPA attack in the firmware update process. According to their work, we deduced the complete



(a)

(b)

Fig.1. CCM mode. (a) CCM encryption mode. (b) CCM decryption mode.

firmware update scheme for Philips Hue smart lamps, which is shown in Fig.2. As a type of IoT terminal device, the firmware update transmission process for Philips Hue smart lamps can refer to Appendix A. In this subsubsection, we introduce the loading process for the firmware update of Philips Hue smart lamps.

In Philips Hue smart lamps, the application layer and the bootloader work together to implement the firmware update. After receiving the new firmware from the IoT cloud service, the application layer stores the new firmware packet in off-chip Flash memory. Then, the application layer sets a flag to indicate that there is new firmware in the off-chip Flash and performs a software reset to restart the MCU. The bootloader checks the update flag of firmware. If the update flag is set, the bootloader verifies the firmware data. If the verification is successful, the bootloader erases the Flash memory stored the application layer and loads the new firmware into it. Finally, the bootloader clears the update flag of firmware to indicate that firmware update is successful.

In the firmware update process, security issues may exist in two phases: network transmission and firmware loading into the IoT terminal device. In the network transmission phase, the security of the firmware update is guaranteed by the transport protocol and update framework [9, 12–14]. In the firmware loading into the IoT terminal device phase, the AE scheme is introduced into the firmware update bootloader program. However, the SCPA attack is a serious threat to the process of decrypting and authenticating the new firmware in the bootloader [8]. Hence, designing a bootloader that can effectively defend against SCPA attacks with low cost can improve the security of firmware update in IoT ecosystem.

2.3 Breaking Philips' Bootloader with SCPA

AES-CCM is used in the bootloader of Philips Hue smart lamps to protect the firmware update. Ronen *et al.* [8] used a novel SCPA to recover the key and nonce value used in AES-CCM.

A cryptographic device consumes power when performing an arbitrary operation. A power trace is a set of power consumption measures during a cryptographic operation. Different operations produce different power traces. With these power traces, it is possible to determine certain secret intermediate values processed in the cryptographic device. This attack method was first demonstrated as differential power analysis (DPA) by Kocher *et al.* [21]. DPA divides a number of power traces into two different sets. By subtracting the value of these sets, the true value of an intermediate bit of operands can be determined. Correlation power analysis (CPA), proposed by Brier *et al.* [22], uses a more complex leakage assumption. Instead of determining a single bit at a time, a CPA attack makes it possible to rapidly recover the value of a byte. Ronen *et al.* [8] used the byte-wise CPA method to recover the key and nonce value of AES-CCM in the SCPA attack.

In the firmware update process, the attacker focuses

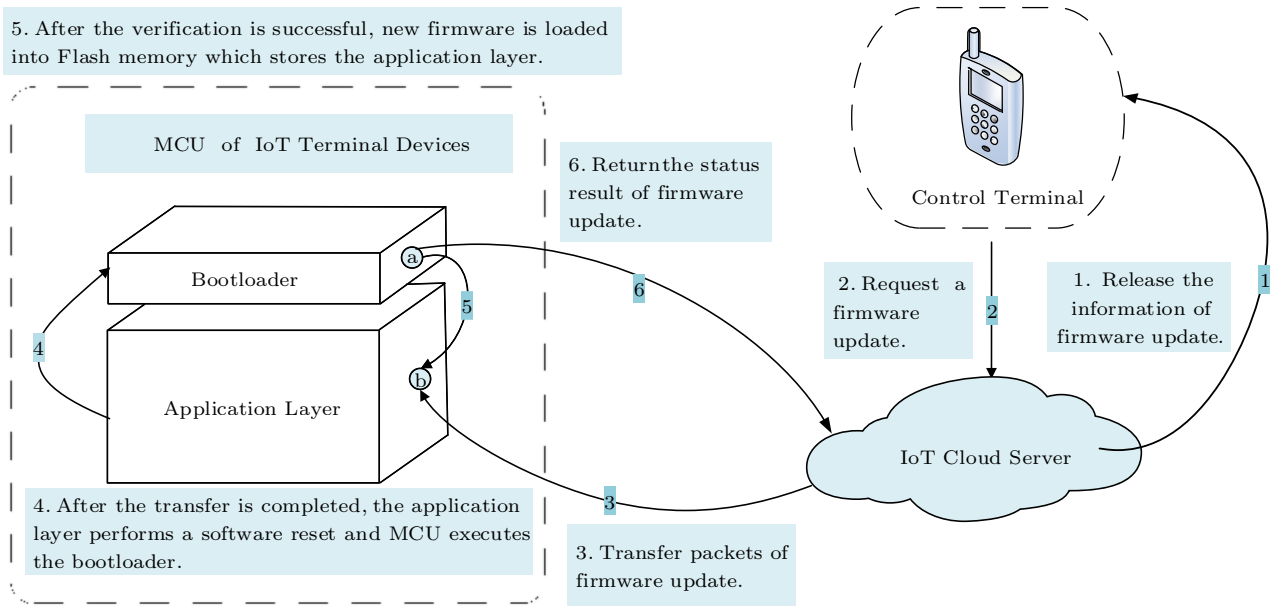


Fig.2. Firmware update scheme of Philips Hue smart lamps.

on the verification phase of ciphertext blocks C_1 and C_2, which can be seen in Fig.3. In the bootloader attack of the Philips Hue smart lamps, $A_1, A_2, \ldots, A_a$ and Ctr_1 must be fixed as constants, which leads S_1 and T_a to be constant.

For the ciphertext block C_1,, the interest attack points focus on SubBytes operations in the first and the second AES rounds. The intermediate value sb_1 of the first round of C_1 can be written as:

$$sb_1 = SB(S_1 \oplus T_a \oplus K \oplus C_1) = SB(K' \oplus C_1),$$

where K is the key of AES-CCM and $K' = S_1 \oplus T_a \oplus K$. During the SCPA attack process, the attacker chooses C_1 to trigger the firmware update, and acquires power traces. Then, the attacker guesses K' to calculate sb_1, and uses the correlation between the power traces and sb_1 to recover K' by the correct guess. K' can be recovered by repeating the aforementioned correlation analysis 16×256 times.

The intermediate value sb_2 of the second round of the C_1 ciphertext block can be written as:

$$\begin{aligned} sb_2 &= SB(MC \circ SR \circ SB(K' \oplus C_1) \oplus K_1) \\ &= SB(f(C_1 \oplus K') \oplus K_1), \end{aligned}$$

where K_1 is the subkey of the second round of the C_1 ciphertext block. After recovering K', the attacker can use the power traces acquired by changing the value of C_1 to recover K_1 by the CPA method. Thus, the key K of AES-CCM can be deduced by K_1 using the inverse key schedule function.

Now, the attack can be repeated for the first round of the C_2 ciphertext block. $T_{P1} = E_K(K' \oplus C_1)$; when C_1 is fixed, T_{P1} is a constant. Focusing on the intermediate value sb_3, the attacker chooses the value of C_2 to trigger the firmware update and acquires power traces. Then, the attacker guesses S_2 to calculate sb_3, and uses the correlation between the power traces and sb_3 to recover S_2 by the correct guess. S_2 can be recovered by repeating the aforementioned correlation analysis 16×256 times. Finally, the nonce value N included in Ctr_2 can be deduced by decrypting S_2 with the key K, i.e., $Ctr_2 = C(N, 2) = E_K^{-1}(S_2)$.

To successfully recover the key and nonce of AES-CCM in the Philips Hue smart lamps using the CPA method, approximately 5 000 power traces for each block must be acquired, and the associated data in the firmware in the acquiring process must be the same.

3 New IoT Firmware Update Scheme Against SCPA and DoS Attacks

Inspired by the work of breaking AES-CCM in the bootloader of Philips Hue smart lamps using SCPA [8], we propose a new IoT firmware update scheme against SCPA and DoS attacks that depends primarily on the design of the bootloader and application layer.

The new IoT firmware update scheme is displayed in Fig.4. Compared with the firmware update scheme of Philips Hue smart lamps depicted in Fig.2, the proposed scheme adds two tables: T_{ad} and T_{v}, and ope-

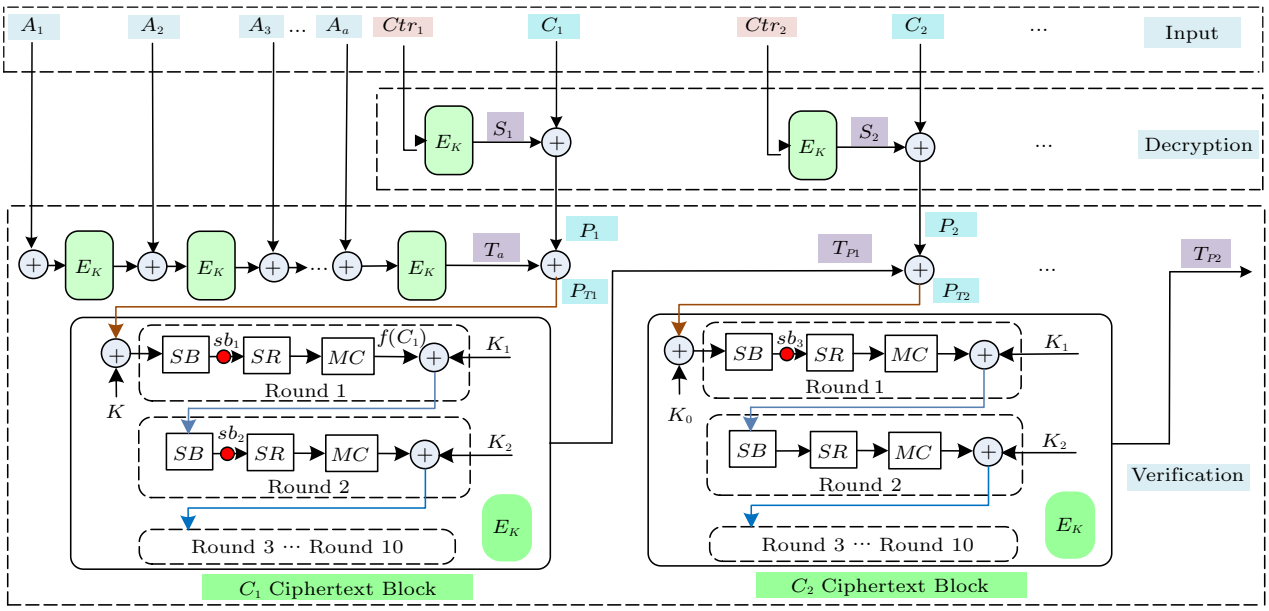


Fig.3. CPA against CCM mode.

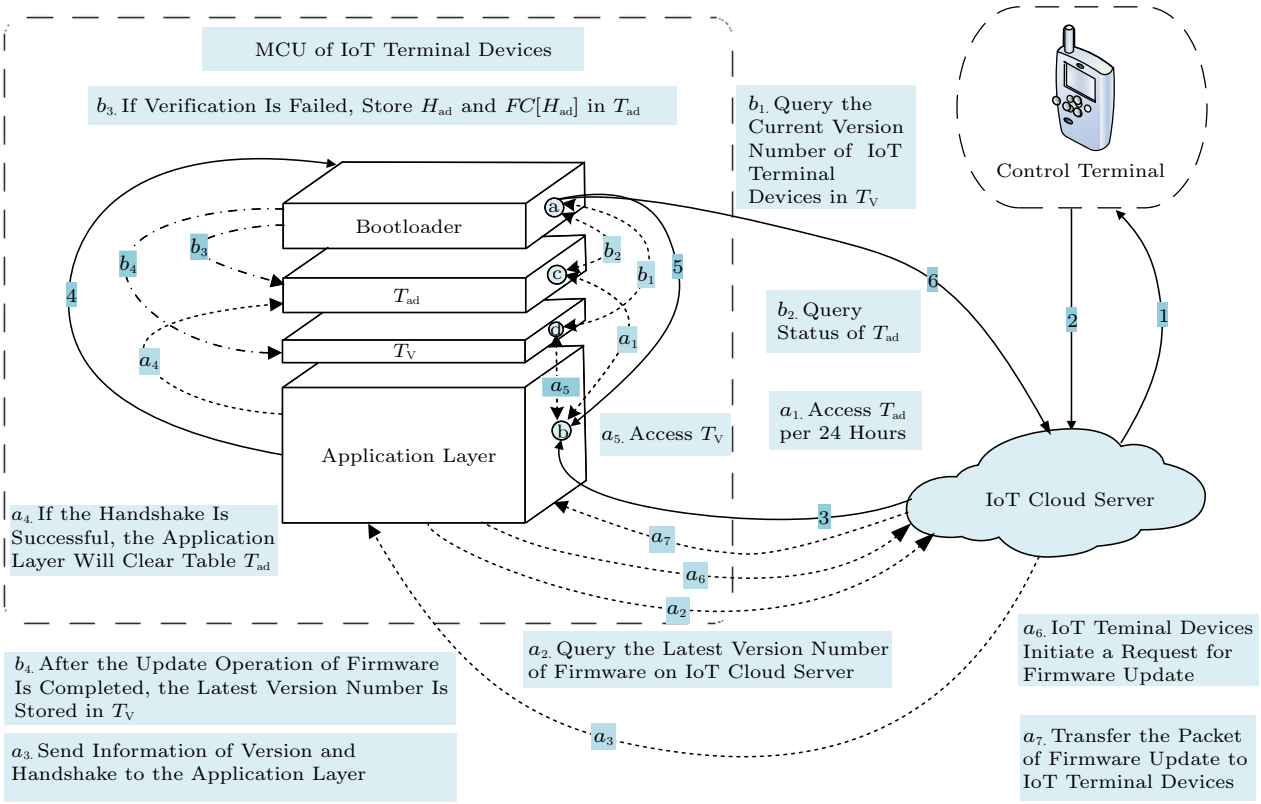


Fig.4. New IoT firmware update scheme against SCPA and DoS attacks. Steps 1, 2, 3, 4, 5 and 6 have same meaning as corresponding steps in Fig.3.

rations: b_i and a_j ($i \in \{1,2,3,4\}$, $j \in \{1,2,\ldots,7\}$). T_{ad} stores the hash value of the associated data in the new firmware; T_{v} stores the current firmware version of the IoT terminal device. b_i ($i \in \{1,2,3,4\}$) is a newly added operation used to interact with T_{ad} and T_{v} in the bootloader. a_j ($j \in \{1,2,\ldots,7\}$) is a newly-added operation used to interact with T_{ad}, T_{v}, and the IoT cloud server in the application layer.

3.1 New Cryptographic Design Against SCPA Attacks in the Bootloader

The core of the proposed scheme against SCPA attacks is to use a technique to limit the maximum number of authentication times (i.e., update times) per unit time. Firstly, some notations about cryptographic design against SCPA attacks are given. In the proposed scheme, T_{ad} is predefined to store the different hash values H_{ad} of the associated data and a counter of failed firmware verifications $FC[H_{\text{ad}}]$. When there are five different H_{ad} in table T_{ad}, we say that T_{ad} is full. That is, the upper bound for the number of H_{ad} and the value $FC[H_{\text{ad}}]$ is five in the proposed scheme. However, the product manufacturer can set the upper bound according to the specific implementation.

Secondly, the workflow of bootloader is shown as follows. The bootloader first reads the update flag of the firmware $UpFlag$ and verifies if $UpFlag$ is set. If $UpFlag$ is set, the bootloader performs the firmware update operations. The operation flow of the new firmware update scheme in the bootloader is described as follows.

b_1. Query T_{v} to obtain the current version $VersionInMCU$ of firmware in the IoT terminal device. Check if the version number $VersionRecieved$ of the new firmware is greater than $VersionInMCU$. If $VersionRecieved > VersionInMCU$, go to step b_2. Otherwise, go to step b_5.

b_2. Query the status of T_{ad}.

b_{2_1}. Check if T_{ad} is full. If T_{ad} is full, go to step b_5. Otherwise, read the associated data in the new firmware and calculate the hash value H_{ad} of the associated data with a hash function such as SHA-256.

b_{2_2}. Check if H_{ad} is equal to the previous hash value, which has been stored in T_{ad}. If they are equal, go to step b_{2_3}. Otherwise, go to step b_{2_4}.

b_{2_3}. Check if $FC[H_{\text{ad}}]$ is greater than or equal to 5. If $FC[H_{\text{ad}}] \geqslant 5$, go to step b_5. Otherwise, go to step b_{2_4}.

b_{2_4}. Read the firmware data and verify. If the verification is successful, the bootloader performs the firmware update operation and then goes to step b_4. Otherwise, go to step b_3.

b_3. Increase $FC[H_{\text{ad}}]$ by 1, i.e., $FC[H_{\text{ad}}] = FC[H_{\text{ad}}] + 1$. Store H_{ad} and $FC[H_{\text{ad}}]$ in T_{ad}. Go to step b_5.

b_4. Store the latest version number in T_{v}.

b_5. Clear the update flag of the firmware $UpFlag$. The bootloader completes.

In the bootloader, operations b_2 and b_3 are used to limit the number of updates, and operations b_1 and b_4 are used to ensure that the version of the firmware update is the latest.

The bootloader queries the status of T_{ad} in step b_2. If T_{ad} is full, the bootloader does not perform the firmware update operations until T_{ad} is cleared by the application layer. Note that T_{ad} and T_{v} are stored in the on-chip Flash which prevents the attacker from modifying stored data by the security fuses. The process of clearing T_{ad} is accomplished by IoT terminal devices by successfully handshaking with the IoT cloud server every 24 hours. Thus, the attacker can obtain no more than 25 power traces every 24 hours. Further, only five of these power traces can be used to deduce the secret cryptographic elements by the SCPA method proposed in [8]. If the attacker attempts to use the SCPA method to recover the key and nonce of AES-CCM in the bootloader, it would require an extremely long time to acquire a sufficient number of useful power traces. Hence, the new firmware update scheme in the bootloader can effectively defend against SCPA attacks.

3.2 New Access Design Against DoS Attacks in the Application Layer

A DoS attack could exist in the previous and proposed schemes. In the proposed scheme, if T_{ad} is full, although the correct firmware update packet sent by the IoT cloud server is received, the firmware update operation is not performed, which leads to a DoS attack. To prevent a DoS attack, new access operations and a handshake protocol are designed and applied to the application layer.

The new access design against DoS attacks in the application layer is displayed in Fig.5, and its specific steps are described as follows.

a_1. The application layer accesses T_{ad} every 24 hours.

a_{1_1}. The application layer queries the status of T_{ad} every 24 hours.

a_{1_2}. T_{ad} returns the status to the application layer. If T_{ad} is full, go to step a_2.

a_2. The application layer queries the version number of the latest firmware update on the IoT cloud server through a new handshake protocol.

a_3. The IoT cloud server returns the information for the latest version and handshake to the application layer of IoT terminal devices. The application layer checks if the handshake is successful. If the handshake is successful, the latest firmware version $VersionRecieved$ from the packet returned by the IoT cloud server is obtained.

a_4. The application layer clears T_{ad} to enable the bootloader to repeat the firmware update process. When T_{ad} is cleared, the firmware update operation can be initiated by IoT terminal devices or the control

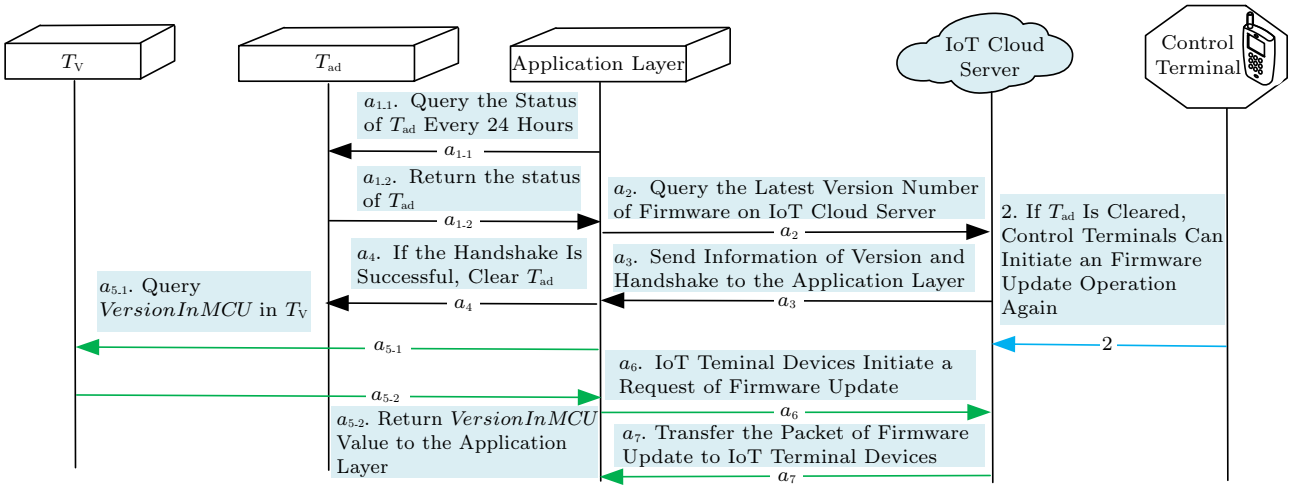


Fig.5. Access design against DoS attacks in the application layer.

terminal APP.

a_5. The application layer accesses $T_{\rm v}$.

a_{5_1}. The application layer queries the current version number $VersionInMCU$ in $T_{\rm v}$.

a_{5_2}. $T_{\rm v}$ returns the current version number $VersionInMCU$ of IoT terminal devices to the application layer. If $VersionRecieved$ is greater than $VersionInMCU$, go to step a_6. Otherwise, the application layer waits for update operations initiated by the control terminal, i.e., step 2 in Fig.5.

a_6. The application layer of IoT terminal devices initiates a request for firmware update to obtain the latest firmware update.

a_7. The IoT cloud server sends the firmware update packet to the application layer of IoT terminal devices. The application layer stores the firmware update packet in the off-chip Flash.

Proposed Handshake Protocol. The proposed new handshake protocol is used to confirm that the version information and handshake are sent by the IoT cloud server. The application layer can only clear $T_{\rm ad}$ if the handshake is successful. Otherwise, $T_{\rm ad}$ will remain unchanged because the IoT terminal device could be under attack.

The specific process of the handshake protocol is described as follows.

1) The application layer generates a random number $RandomN$ and stores it for subsequent verification operations. $RandomN$ is encrypted with a cipher such as AES to obtain the encrypted data $CipherRan$. The application layer shares the secret key of the cipher with the IoT cloud server. The command data for version query and $CipherRan$ as a version query packet is sent to the IoT cloud sever by the application layer.

2) After receiving the version query packet, the IoT cloud server uses the same cipher with the application layer to decrypt $CipherRan$ to obtain $RandomN$. Then, the IoT cloud server encrypts $(RandomN+1)$ to obtain $CipherRanS$. The IoT cloud server sends $CipherRanS$ and the latest version data $VersionRecieved$ as a request response packet to the IoT terminal device.

3) After receiving the request response packet, the IoT terminal device decrypts $CipherRanS$ to obtain $HandShakeValue$. If $HandShakeValue$ is equal to $(RandomN+1)$, the handshake operation is successful and the application layer can clear $T_{\rm ad}$. Otherwise, $T_{\rm ad}$ leaves the status unchanged.

This new handshake protocol can effectively prevent the attacker from impersonating the IoT cloud server and sending the handshake information to IoT terminal devices.

According to Subsection 2.3, we know that the condition for successfully breaking the AES-CCM in the bootloader is that for the given C_1 ciphertext block, T_a remains constant while acquiring approximately 5 000 traces in a few days [8]. In the proposed firmware update scheme, the operations a_1, b_1, and b_3 work together to limit the number of update times to no more than 25 every 24 hours. Moreover, only five power traces in the 25 updates can be used in the SCPA attack. If T_a remains constant, acquiring 5 000 traces would require approximately three years. This excessive time cost should prevent most attackers from using the SCPA attack to recover the key of AES-CCM in the bootloader. Operations a_2, a_3, and a_4 work together to clear $T_{\rm ad}$, which then enables the bootloader to regain firmware update permission. Thus, we can effectively defend against DoS attacks.

4 Implementation

The common hardware scheme for IoT terminal devices is: MCU + off-chip Flash. The MCU is used primarily to store the bootloader and application layer. The off-chip Flash is used mainly to store the firmware update data. The firmware update schemes of the Philips Hue smart lamp and the proposed modified scheme were implemented on a test platform based on STM32F405+M25P40, which is widely used in IoT terminal devices.

4.1 Test Platform

The test platform for evaluating the firmware update scheme is displayed in Fig.6. The equipment and tools used in the test platform include: PC, RT809 programmer, J-link simulator, external Flash board, STM32F405 board, and oscilloscope.

The operation process of the test platform is described as follows.

1) The AES-CCM based firmware encryption program is implemented on the PC. The encryption program reads the code data $Code.bin$ to perform an encryption operation.

2) The encryption program performs an encryption operation to generate the encrypted firmware data and writes it to firmware file $AEfirmware.bin$.

3) The RT809 programmer loads $AEfirmware.bin$ to the external Flash board.

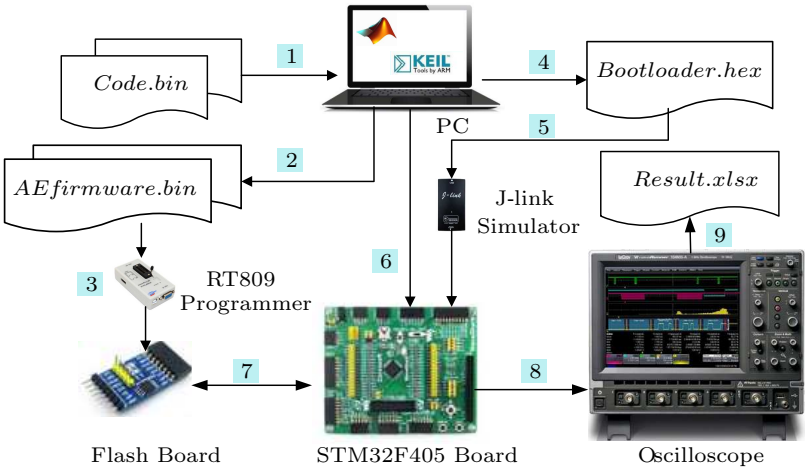


Fig.6. Test platform for evaluating firmware update scheme.

4) The bootloader firmware update program based on the STM32F405 hardware platform is implemented, compiled, and linked to generate the executable HEX file *Bootloader.hex* in $KEILuVision5$.

5) The *Bootloader.hex* file is loaded to the STM32F405 board through the J-link simulator.

6) To facilitate the test, the PC sends a command to the STM32F405 board through the serial port to trigger the firmware update operation.

7) During the firmware update process, the STM32F405 reads and writes data from the external Flash board.

8) During the firmware update process, the TM32F405 controls the high or low levels of the GPIOB1. The oscilloscope monitors the output level of the GPIOB1 and provides the timing waveform.

9) *Result.xlsx* is generated according to the timing waveform acquired by the oscilloscope.

4.2 Test Results

In order to compare the memory cost and update the efficiency of the proposed scheme with Philips' scheme, two firmware update schemes in the bootloader are implemented and tested. The Philips Hue smart lamps' scheme corresponding to Fig.2 cannot defend against SCPA. The proposed firmware update scheme corresponding to Fig.4 can effectively counter SCPA attacks.

4.2.1 Memory Cost

The Philips Hue smart lamps' scheme requires 18.05 KB of Flash memory and 17.86 KB of SRAM. The proposed scheme requires 20.4 KB of Flash memory and 17.88 KB of SRAM. The RAM cost of both schemes is virtually the same. Compared with Philips, the Flash memory of the proposed scheme is increased by 2.35 KB. The test results of the Flash memory cost are given in Table 2. The additional Flash memory used in the proposed scheme is due to the operations for H_{ad} and T_{v}, and the SHA-256 library function, which calculates the hash value of the associated data in the firmware update.

Table 2. Test Results of Flash Memory Cost (KB)

Type	Fupdate	Crypto	Hdrives	Total
Philips	1.55	10.58	5.92	18.05
Proposed	2.02	12.22	6.16	20.40

Note: Fupdate denotes bootloader code for firmware update. Crypto denotes STM32 cryptographic library. Hdrives denotes hardware drivers code.

The data stored in Flash memory include the bootloader code for the firmware update, STM32 cryptographic library[6], and hardware driver code. In Table 2, the Fupdate of the proposed scheme is 0.47 KB greater than Philips because of the checking and storing operations of H_{ad} and T_{v}. The bootloader code for the firmware update in the proposed scheme requires only 2.02 KB of Flash memory. Because the bootloader implementation cost of the proposed scheme is very low, it can be applied to a variety of resource-constrained IoT terminal devices.

Compared with Philips, the Crypto for the proposed scheme is increased by 1.64 KB because of the call to the SHA-256 interface function. The size of Flash memory

[6]https://www.st.com/zh/embedded-software/stm32-cryp-lib.html, Sept. 2013.

required for Crypto depends on the type of hardware platform and the number of interface functions used in IoT terminal devices. To optimize the implementation cost of the cryptographic algorithm, IoT terminal devices manufacturer can choose MCUs with pure hardware cryptographic accelerators.

4.2.2 *Update Efficiency*

The firmware update process for the bootloader includes four main phases: 1) verification phase, 2) on-chip Flash block erasing phase, 3) decryption phase, and 4) writing code into the on-chip Flash phase. The update rate of different firmware updates was tested on the STM32F405+M25P40 hardware platform. Further, to study the effect of the firmware length on the update time, we provided nine firmware updates with different lengths. The time and rate test results of the firmware update in Philips and the proposed scheme are listed in Table 3. Compared with Philips, the range of additional update time required in the proposed scheme was 0.05 s–0.32 s, and the range of reduction in update rate was 0.06 KB/s–0.46 KB/s.

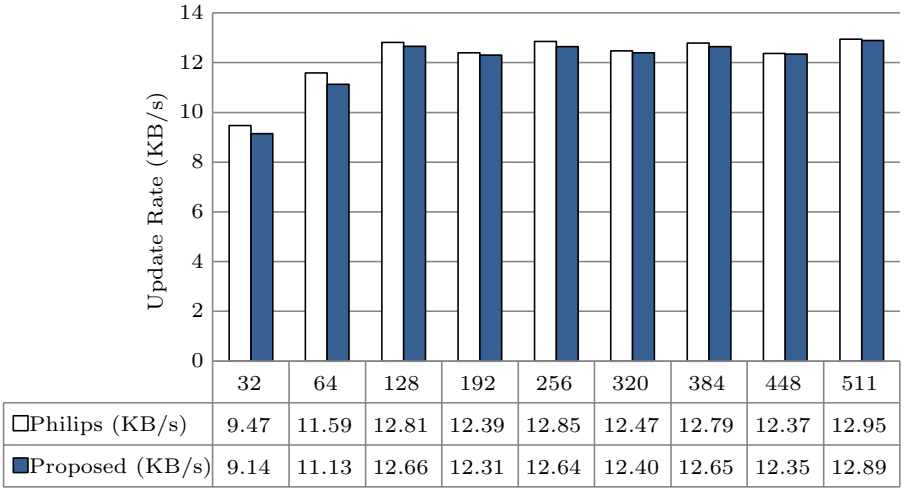
To clearly illustrate the difference of update rate between Philips and the proposed scheme, the comparison results are displayed in the form of a histogram in Fig.7. Compared with Philips, the update rate of the proposed scheme was not significantly reduced. The update rate of the proposed scheme can satisfy the actual demand of the product. Hence, the proposed firmware update scheme in the bootloader can be used in IoT terminal devices.

The time ratios of the different phases were different based on the firmware length. The time ratio histogram of the four phases in different firmware in the proposed scheme is displayed in Fig.8. With the increase of firmware length, the time in the verification and decryption phases increased. For larger size firmware, the verification and the decryption time became the main factor influencing the firmware update rate. Therefore, to improve the update rate of large size firmware, it would be a better choice for the IoT manufacturer to choose MCUs with pure hardware cryptographic accelerators in the hardware design of their IoT product.

Table 3. Time and Rate of Firmware Update

Size (KB)	Type	VT (s)	ET (s)	DT (s)	WT (s)	TT (s)	UE (KB/s)
32	Philips	0.90	1.13	0.86	0.49	3.38	9.47
	Proposed	0.91	1.24	0.86	0.49	3.50	9.14
64	Philips	1.81	1.01	1.72	0.98	5.52	11.59
	Proposed	1.80	1.25	1.73	0.97	5.75	11.13
128	Philips	3.61	1.00	3.45	1.93	9.99	12.81
	Proposed	3.61	1.08	3.45	1.97	10.11	12.66
192	Philips	5.41	1.96	5.20	2.93	15.50	12.39
	Proposed	5.42	2.11	5.18	2.89	15.60	12.31
256	Philips	7.22	1.94	6.88	3.89	19.93	12.85
	Proposed	7.22	2.18	6.93	3.92	20.25	12.64
320	Philips	9.02	3.10	8.64	4.90	25.66	12.47
	Proposed	9.03	3.33	8.64	4.80	25.80	12.40
384	Philips	10.83	3.02	10.42	5.76	30.03	12.79
	Proposed	10.83	3.22	10.08	6.22	30.35	12.65
448	Philips	12.63	4.44	12.01	7.14	36.23	12.37
	Proposed	12.63	4.60	12.04	7.01	36.28	12.35
511	Philips	14.40	3.64	13.67	7.76	39.47	12.95
	Proposed	14.42	3.82	13.86	7.54	39.64	12.89

Note: VT denotes verification time. ET denotes on-chip Flash block erasing time. DT denotes decryption time. WT denotes code writing into on-chip Flash time. TT denotes total time. UE denotes firmware update rate.



	32	64	128	192	256	320	384	448	511
Philips (KB/s)	9.47	11.59	12.81	12.39	12.85	12.47	12.79	12.37	12.95
Proposed (KB/s)	9.14	11.13	12.66	12.31	12.64	12.40	12.65	12.35	12.89

Fig.7. Comparison of update rate between Philips and the proposed scheme.

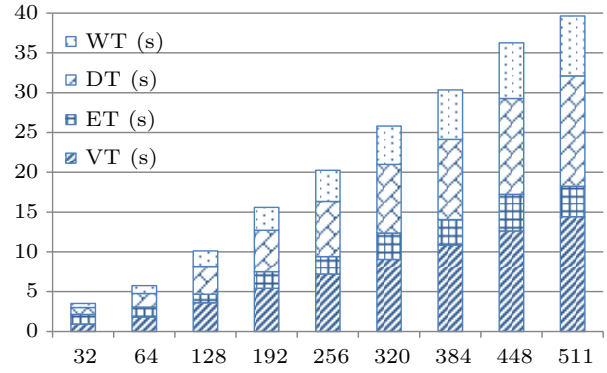


Fig. 8. Time ratio of four phases in different firmware in the proposed scheme.

5 Conclusions

In this paper, we proposed a new firmware update scheme against SCPA and DoS attacks for IoT terminal devices. From the system and algorithm level perspective, this firmware update scheme uses countermeasures with lower implementation cost to guarantee the security in the firmware update process. In the bootloader, the operations of checking and storing T_{ad} are added to effectively defend against SCPA attacks by preventing the attacker from acquiring a sufficient number of useful traces in a short time. In the application layer, the handshake protocol and access operations for T_{ad} and T_v are added to enable the bootloader to regain update permission, which can effectively defend against DoS attacks.

The proposed firmware update scheme can improve the security of the new firmware loading process at extremely low cost, which has reasonable engineering tradeoffs in terms of security and cost. Hence, in addition to Philips Hue lamps, it can also be applied to other resource-constrained IoT terminal devices to guarantee the security of firmware updates.

For future work, a lightweight and secure AE scheme will be designed and applied into the loading process of IoT firmware update to further reduce the cost of implementation and improve update efficiency. We hope the proposed scheme could be applied in IoT terminal devices to enhance the security of IoT ecosystem.

References

[1] Li W, Song H, Zeng F. Policy-based secure and trustworthy sensing for Internet of things in smart cities. *IEEE Internet of Things Journal*, 2018, 5(2): 716-723. DOI: 10.1109/JIOT.2017.2720635.

[2] Patton M, Gross E, Chinn R *et al.* Uninvited connections: A study of vulnerable devices on the internet of things (IoT). In *Proc. the 2014 IEEE Joint Intelligence and Security Informatics Conference*, Sept. 2014, pp.232-235. DOI: 10.1109/JISIC.2014.43.

[3] Antonakakis M, April T, Bailey M *et al.* Understanding the Mirai Botnet. In *Proc. the 26th USENIX Security Symposium*, Aug. 2017, pp.1093-1110.

[4] Kim J, Chou P H. Energy-efficient progressive remote update for flash-based firmware of networked embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 2010, 16(1): Article No. 7. DOI: 10.1145/1870109.1870116.

[5] Wurm J, Hoang K, Arias O *et al.* Security analysis on consumer and industrial IoT devices. In *Proc. the 21st Asia and South Pacific Design Automation Conference*, Jan. 2016, pp.519-524. DOI: 10.1109/ASPDAC.2016.7428064.

[6] Radanliev P, De Roure D, Cannady S *et al.* Economic impact of IoT cyber risk — Analysing past and present to predict the future developments in IoT risk analysis and IoT cyber insurance. In *Proc. the 2018 Living in the Internet of Things: Cybersecurity of the IoT*, Mar. 2018, Article No. 3. DOI: 10.1049/cp.2018.0003.

[7] Cui A, Costello M, Stolfo S. When firmware modifications attack: A case study of embedded exploitation. In *Proc. the 20th Annual Network and Distributed System Security Symposium*, Feb. 2013. DOI: 10.7916/D8P55NKB.

[8] Ronen E, Shamir A, Weingarten A O, O'Flynn C. IoT goes nuclear: Creating a ZigBee chain reaction. In *Proc. the 2017 IEEE Symposium on Security and Privacy*, May 2017, pp.195-212. DOI: 10.1109/SP.2017.14.

[9] Idrees M S, Schweppe H, Roudier Y *et al.* Secure automotive on-board protocols: A case of over-the-air firmware updates. In *Proc. the 3rd Int. Workshop. Communication Technologies for Vehicles*, Mar. 2011, pp.224-238. DOI: 10.1007/978-3-642-19786-4_20.

[10] Steger M, Karner M, Hillebrand J *et al.* Applicability of IEEE 802.11s for automotive wireless software updates. In *Proc. the 13th International Conference on Telecommunications*, Jul. 2015. DOI: 10.1109/ConTEL.2015.7231190.

[11] Prada-Delgado M A, Vázquez-Reyes A, Baturone I. Trustworthy firmware update for Internet-of-Thing devices using physical unclonable functions. In *Proc. the 2017 Global Internet of Things Summit*, Jun. 2017. DOI: 10.1109/GIOTS.2017.8016282.

[12] Choi B C, Lee S H, Na J C, Lee J H. Secure firmware validation and update for consumer devices in home networking. *IEEE Transactions on Consumer Electronics*, 2016, 62(1): 39-44. DOI: 10.1109/TCE.2016.7448561.

[13] Yohan A, Lo N W. An over-the-blockchain firmware update framework for IoT devices. In *Proc. the 2018 IEEE Conference on Dependable and Secure Computing*, Dec. 2018. DOI: 10.1109/DESEC.2018.8625164.

[14] Lee B, Lee J H. Blockchain-based secure firmware update for embedded devices in an Internet of Things environment. *The Journal of Supercomputing*, 2017, 73(3): 1152-1167. DOI: 10.1007/s11227-016-1870-0.

[15] Asokan N, Nyman N, Rattanavipanon N *et al.* ASSURED: Architecture for secure software update of realistic embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018, 37(11): 2290-2300. DOI: 10.1109/TCAD.2018.2858422.

[16] O'Flynn C, Chen Z. Side channel power analysis of an AES-256 bootloader. In *Proc. the 28th IEEE Canadian Conference on Electrical and Computer Engineering*, May 2015, pp.750-755. DOI: 10.1109/CCECE.2015.7129369.

[17] Guillen O M, De Santis F, Brederlow R, Sigl G. Towards side-channel secure firmware updates. In *Proc. the 9th Int. Symp. Foundations and Practice of Security*, Oct. 2016, pp.345-360.

[18] Dworkin M. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. Technical Report, National Institute of Standards and Technology, 2004. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication8-00-38c.pdf, Dec. 2019.

[19] IEEE. IEEE Standard for Information technology—Telecommunications and information exchange between systems — Local and metropolitan area networks — Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11*, Jul. 2004.

[20] Dworkin M. Recommendation for block cipher modes of operation: Methods and techniques. Technical Report, National Institute of Standards and Technology, 2001. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf, Dec. 2019.

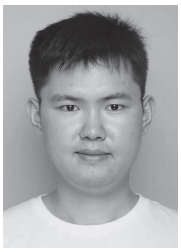
[21] Kocher P, Jaffe J, Jun B. Differential power analysis. In *Proc. the 19th Annual Int. Cryptology Conf.*, Aug. 1999, pp.388-397. DOI: 10.1007/3-540-48405-1_25.

[22] Brier E, Clavier C, Olivier F. Correlation power analysis with a leakage model. In *Proc. the 6th International Workshop on Cryptographic Hardware and Embedded Systems*, Aug. 2004, pp.16-29. DOI: 10.1007/978-3-540-28632-5_2.

Yan-Hong Fan received her M.S. degree in detection technology and automation device from the University of Electronic Science and Technology of China, Chengdu, in 2006. She is currently pursuing her Ph.D. degree in the School of Cyber Science and Technology from Shandong University, Qingdao. Her main research focuses on the analysis of symmetric ciphers and side-channel analysis.

Mei-Qin Wang received her Ph.D. degree in information security from School of Mathematics, Shandong University, Jinan, in 2007. Since 2007, she has been with Shandong University. Currently, she is a professor in the School of Cyber Science and Technology from Shandong University, Qingdao. Her research focuses on cryptography science and technology, in particular, the design and analysis of symmetric ciphers. She received the First-Class Award of National Cryptography Scientific and Technological in 2017 and the Second-Class Award of Cryptographic Innovation of China Cryptographic Society in 2014.

Yan-Bin Li received her M.S. degree in information security from Shandong University, Jinan, in 2015. She is currently pursuing her Ph.D. degree in information security from the same university. Her main research focuses on the design and analysis of symmetric ciphers, in particular, authenticated ciphers.



Kai Hu received his B.S. degree in information security from Shandong University, Jinan, in 2016. He is currently pursuing his Ph.D. degree in the School of Cyber Science and Technology from Shandong University, Qingdao. His main research focuses on the analysis of symmetric ciphers.

Mu-Zhou Li received his B.S. degree in information security from Shandong University, Jinan, in 2017. He is currently pursuing his Ph.D. degree in the School of Cyber Science and Technology from Shandong University, Qingdao. His main research is the analysis of symmetric ciphers.

Appendix: Typical IoT Firmware Update Process

Fig.A.1 shows a typical network topology for an IoT firmware update which includes the following entities.

1) *Product Manufacturer* (*PM*). The PM generates a firmware update packet for IoT terminal devices and uploads it to the IoT cloud server.

2) *Control Terminal* (*CT*). The CT (e.g., Smartphone, IPad, PC) receives the firmware update information released by the IoT cloud server and remotely monitors and updates IoT terminal devices by APP.

3) *IoT Cloud Service* (*CS*). The IoT CS receives and stores the firmware update packets from the PM and modifies the version number of the firmware update. After receiving a request for a firmware update, the IoT CS sends the firmware update packet to IoT terminal devices.

4) *Wide Area Network* (*WAN*). The WAN uses the Ethernet protocol to transmit the IoT firmware data. The WAN connects with the IoT CS and the LAN gateway controller.

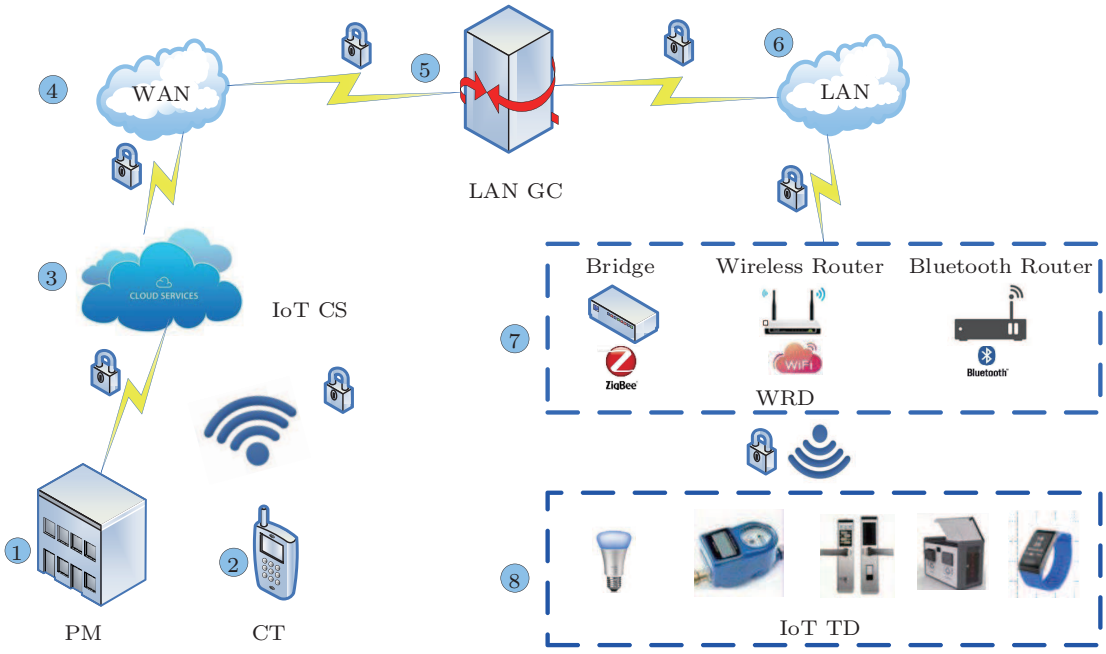


Fig.A.1. Typical network topology of IoT firmware update.

5) *Local Area Network Gateway Controller* (*LAN GC*). The LAN GC acts as a data hub to process the data from or to IoT terminal devices. LAN GC connects with the WAN and LAN.

6) *Local Area Network* (*LAN*). It uses the LAN protocol to transmit the IoT firmware data, and connects with the LAN GC and wireless relay devices.

7) *Wireless Relay Device* (*WRD*). The WRD (e.g., bridge, wireless router and bluetooth router) uses the wireless communication protocols (e.g., Bluetooth, Wi-Fi, and Zigbee) to send the firmware update to IoT terminal devices.

8) *IoT Terminal Devices* (*TDs*). The IoT TDs include different types of environmental monitors (e.g., temperature, humidity, pressure), control actuators (e.g., automobile, industrial production), smart home equipment (e.g., smart lights, smart locks), and wearable devices (e.g., smart bracelets, smart watches). The IoT TDs connect with the WRD by a wireless network port (e.g., Wi-Fi, Bluetooth and Zigbee). In the micro controller unit (MCU) of the IoT TD, the application layer and bootloader are used to implement the function of the firmware update. The application layer receives the firmware update from the IoT CS, and modifies and stores the information for the firmware update in the off-chip Flash. The bootloader first checks the firmware parameter information. If there is new firmware, it verifies and moves the new firmware from the off-chip Flash to the on-chip Flash.

The typical firmware update process initiated by CT APP is described as follows.

1) The PM ① sends the firmware update to the IoT CS ③ for network release.

2) After receiving the release information for the firmware update from the IoT CS ③, the CT APP ② sends a firmware update command to the IoT CS ③.

3) Controlled by the CT APP ②, the IoT CS ③ sends the firmware update packet to the IoT TD ⑧ through the WAN ④, LAN GC ⑤, LAN ⑥, and WRD ⑦ in turn.

4) After receiving the complete firmware update packet, the IoT TD ⑧ verifies the firmware update. If verification is successful, the IoT TD replaces the current firmware with the new firmware. Otherwise, the IoT TD leaves the current firmware unchanged.

5) The IoT TD ⑧ returns the result of the firmware update to the IoT CS ③ through the WRD ⑦, LAN ⑥, LAN GC ⑤, and WAN ④ in turn.