This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2820086, IEEE Access

A Handshake Protocol with Unbalanced Cost for Wireless Updating

Jiaren Cai, Xin Huang, Jie Zhang, Jiawei Zhao, Yaxi Lei, Dawei Liu, Xiaofeng Ma

***Abstract*—Wireless updating is an essential method to update system files or fix bugs in IoT devices. A significant and challenging problem in wireless updating is security. First, without security guarantees, attackers can utilize the updating procedure to install harmful programs into the victim devices. Second, it is challenging to provide security for wireless updating, since in many IoT scenarios, the devices to be updated are computationally limited devices and located far from the center that issues update files. Currently, there are two types of solution to protect the wireless updating. The first one is the Transport Layer Security (TLS) protocol or Secure Sockets Layer (SSL) protocol that are used by wireless updating schemes for mobile terminals with the following operation systems: Windows, Debian, Android and iOS. Another solution is the ECDH-based handshake in the Software Defined Function (SDF) wireless updating scheme for IoT devices. However, both the two solutions require equal computation tasks on the update file issuing center and the device to be updated. Normally, the former is much powerful than the latter. Therefore, to further address the security problem in wireless updating, we propose a novel solution with unbalanced computation costs on the two parties. In particular, we design an improved ECDH-based handshake protocol for the SDF wireless updating scheme, namely, the unbalanced OpenFunction handshake protocol. The protocol transfers significant computation task from the limited IoT device to the powerful center. The security of the protocol is analyzed. A prototype is realized to test the performance of the protocol. The experiment results show that in the same experimental platform, our protocol is much lightweight than the TLS handshake protocol and SSL handshake protocol.**

***Index Terms*—Internet of Things, Wireless Updating, Secure Sockets Layer, Transport Layer Security, Software Defined Function, Unbalanced OpenFunction handshake**

I. INTRODUCTION

INTERNET of Things (IoT) is the network of a variety of devices that can automatically interact with each other and cooperate with their neighbors to achieve common goals [1]. The IoT technology is considered to be incorporated into the products and servers of a wide range of industry sectors such as healthcare, automotive, manufacturing, consumer electronics and home [2].

In many IoT systems, wireless updating is a significant method for IoT devices to receive system files from remote servers. Through remote system updates, those devices employed in practical application scenes obtain new functions and fix security vulnerabilities and bugs wirelessly.

To our knowledge, there are several wireless updating schemes. Many mainstream laptop operating systems, such as Windows and Debian [3], [4], implement Secure Sockets Layer protocol (SSL) to finish handshake with servers to achieve wireless system updating. Besides, iOS system applies SSL and Android system utilizes Transport Layer Security Protocol (TLS) to furnish device firmware update for their mobile devices [5], [6]. Furthermore, for IoT applications, [7] ensures security of Wireless Multimedia Delivery with error-resilient encoder and decoder, which processes authentication and watermarking in encoder and verification in decoder. In addition, a protocol called OpenFunction [8], [9] has also proposed a feasible scheme based on Software Defined Function (SDF). Unlike [7] employing decoder to reduce cost of code error, OpenFunction authenticates and verifies Message Authentication Code (MAC) to guarantee security of session key agreement. According to this protocol, some IoT devices which act as function stations or local centers are enabled to securely initiate wireless connections with their remote servers to accept the update files. Then, these function stations utilize the update files to reprogram IoT end devices like sensors through many strategies [10], [11], [12], [13], [14], [15] in Wireless Sensor Network (WSN) or Pervasive Social Network (PSN).

In above schemes, the security of update files is guaranteed by their handshake process. Handshake process is an important step in both SSL and TLS, which is to verify certificates, agree on session keys and establish trusted connections between servers and devices. The handshake processes in these schemes require participants engaging on two sides to execute

This work was supported in part by Natural Science Foundation of China under Grant No. 61701418, in part by Natural Science Foundation of China under Grant No. 61701417, in part by the CERNET under Grant NGII20161010, in part by Jiangsu Province National Science Foundation under grant BK20150376, in part by Suzhou Science and Technology Development Plan under grant SYG201516, and in part by XJTLU RDF140243 and RDF150246.

Jiaren Cai, Xin Huang, Jie Zhang, Yaxi Lei, and Dawei Liu are now with the Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, China.

Jiawei Zhao is now with the International Business School Suzhou, Xi'an Jiaotong-Liverpool University, China.

Xiaofeng Ma is now with the Department of Control Science & Engineering, Tongji University, China.

Corresponding author: Xin Huang (Xin.Huang@xjtlu.edu.cn).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2820086, IEEE Access

authentication with equivalent computation tasks. In fact, it is notable that the computing capacity of these two sides, like function stations and servers, are different. Especially in the scenes of OpenFunction protocol, function stations build secure connections with controllers, i.e. servers, by OpenFunction handshake to ensure the authentication of update files. However, most function stations are less powerful than their servers. The OpenFunction protocol neglects this problem and fails to take advantage of powerful servers to achieve higher efficiency in handshake. Moreover, it might overburden some weak function stations. Therefore, in this paper, we aim to increase efficiency of OpenFunction handshake by way of reducing the computational load on function stations.

In this paper, we will propose an improved protocol of the OpenFunction handshake protocol. We consider function stations as middle devices and regard remote servers as controllers. With shifting some computing tasks from middle devices to controllers, the proposed protocol allocates unbalanced computation in authentication of these both sides. In addition, the security of authentication will not be affected by the shift. Thus, this design reduces the computing cost of function stations and minimizes the risk of overload. In addition, we will design a prototype for this protocol. The main contributions of this paper are summarized as follows:

- The design of unbalanced computation on middle devices and controllers is applied in the OpenFunction protocol for the first time. This design reduces the burden of those devices that possess low computing power, so as to improve the protocol's efficiency and avoid to overload middle devices. Moreover, after comparing experiment results, the proposed protocol enables devices to complete handshake much faster than handshake of SSL and TLS. The protocol's total computing time in device and server is also shorter than those of SSL and TLS. Therefore, by using unbalanced computation design, the unbalanced OpenFunction protocol is more efficient than SSL and TLS.
- The protocol improves the handshake in former OpenFunction with fewer steps. Instead of seven communication steps of the predecessor [9] (as shown in Fig.1), this protocol completes the OpenFunction handshake by only four steps. Thus, the protocol can finish handshake faster than its predecessor.
- The Elliptic Curve Cryptography (ECC) algorithm is firstly utilized in SDF framework. This algorithm can ensure the security of authentication during OpenFunction handshake. By employing ECC algorithm, the secret key stored in two sides is much shorter than that of RSA in same encryption strength. Moreover, this protocol requires no secret key before running.

In the rest of this paper, Section 2 illuminates the background about the SDF and the OpenFunction protocol. Section 3 introduces the knowledge related to ECC algorithm and the symbols used in this paper. The unbalanced OpenFunction protocol is described in Section 4. The security analysis of the protocol is demonstrated in Section 5. The performance of the advised protocol is theoretically evaluated by analyzing its

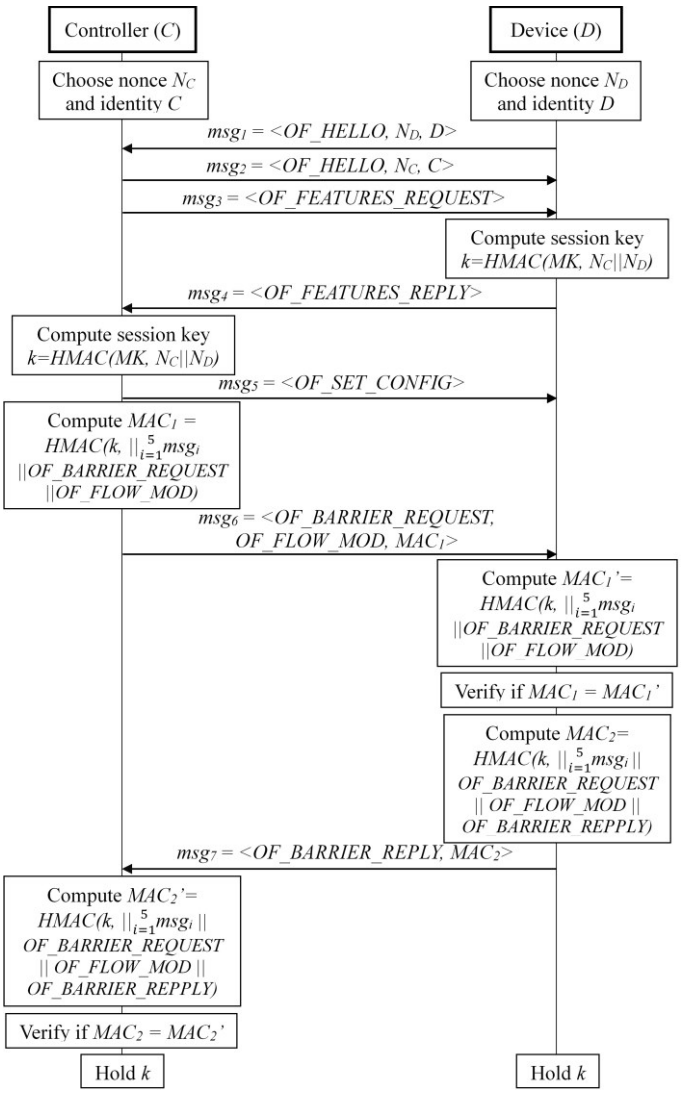


Fig. 1. Handshake process of OpenFunction protocol in SDF [9].

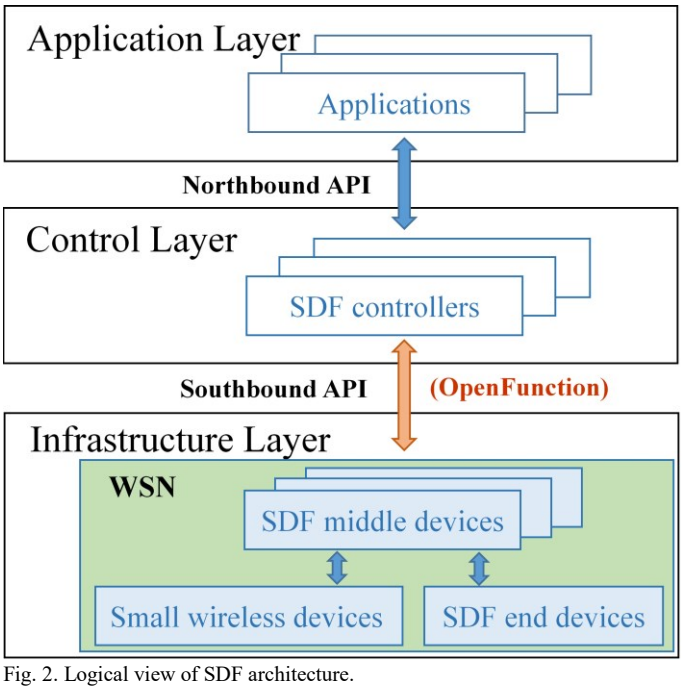


Fig. 2. Logical view of SDF architecture.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2820086, IEEE Access

experiment results in Section 6. The comparison between the proposed protocols and other actual remote system update approaches is listed in Section 7. Furthermore, Section 8 displays a use case of this improved protocol. Finally, the conclusion of this paper is drawn in Section 9.

II. Background

Our work is to improve the OpenFunction handshake protocol in SDF. Therefore, we briefly review SDF and OpenFunction in this section.

A. SDF

Fig.2 shows a logical view of SDF. It is divided into three layers from top to bottom: application layer, control layer, and infrastructure layer. In SDF, the controller in the control layer sends update files to the middle device in the infrastructure layer. With the update files, the middle device reprograms the end devices. Therefore, securing communication between the control and infrastructure layers is a core requirement of SDF. It determines whether the end devices can be reprogrammed properly.

B. OpenFunction

The OpenFunction is a security protocol suit between the control and infrastructure layers. Its security is based on handshake process, which is the OpenFunction handshake protocol in [9]. The OpenFunction handshake protocol authenticates the controller and middle device, and establishes a shared session key between them. The shared key is used to provide security of update files. Fig.1 provides a general overview of this protocol's handshake process.

As we mentioned before, middle device is much less powerful compared with the controller. The OpenFunction handshake protocol requires the same computing cost on them. It is better to reduce the cost on the middle device. In addition, some communications in the protocol are not necessary to handshake. In the following section, we will improve the protocol and design a new handshake protocol that is better than [9] in terms of the following aspects:

- Unbalanced computation tasks in middle device and controller.
- Fewer number of communication steps for handshake.

C. Challenges and Necessities of Unbalanced Computing Cost

Many IoT middle devices have very limited computational power. Executing same computing work requires longer time in these limited devices than in IoT controllers and other powerful devices. When multiple operations execute in a short time, end devices might be influenced by high computational load. The schemes applied unbalanced computing cost can reduce computational burden on the limited devices and shorten runtime.

Furthermore, in some scenarios, many IoT middle devices are only powered by batteries. Unbalanced cost schemes can allocate less computing works on end devices by diverting to powerful devices and demand lower energy cost. Thus, battery replacement frequency of end devices can be longer than usual.

TABLE 1
SYMBOLS AND DEFINITIONS

Symbol	Definition
D, C	Identities of participators. D represents the computationally limited middle device, and C denotes the controller such as a laptop, a powerful mobile phone or a server.
M_i	The message of the ith transmission in the protocol.
R_D, R_C	The random integers selected by D and C respectively
SK_D, SK_C	The ECC private keys held by D and C respectively
PK_D, PK_C	The ECC public keys of D and C derived from private keys and base point G
G	A base point of an elliptic curve E over finite fields
×	The operation of ECC scalar multiplication.
K_{DH}	The provisional Elliptic Curve Diffie-Hellman (ECDH) key. $K_{DH}\_x_1$ is the leftmost 128 bit of X coordinate of K_{DH}, which is used to compute a master key, and $K_{DH}\_x_2$ is the rightmost 128 bit of X coordinate of K_{DH} that is used in authenticating the identity of both sides C and D.
MK	The master key to be generated for further communication.
‖	The concatenation of bit strings.
$HMAC_L(K, M)$	Computing L-bit Hash-based Message Authentication Code (HMAC) for message M under key K.
$CMAC_L(K, M)$	Computing L-bit Cipher-based Message Authentication Code (CMAC) for message M under key K.

Additionally, unbalanced cost schemes lower the requirement of IoT middle devices' computing power in some applications. Users can deploy more lightweight, cheaper and smaller hardware chips in end devices.

III. Preliminaries and Symbols

This section introduces ECC algorithm and security model for the improved protocol. Symbols used in the rest part of this paper are summarized in Table 1.

A. Elliptic Curve Public Key Cryptography

The Diffie-Hellman key exchange [16]66 of the elliptic curve public key cryptography (ECC) is essential in the unbalanced OpenFunction protocol. Here we briefly introduce the related knowledge.

1) Elliptic Curve

Suppose that *(x,y)* is a point on the elliptic curve; *a* and *b* are coefficients; p is an odd prime; and *GF(p)* is a prime finite field.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2820086, IEEE Access

An elliptic curve can be described by the following formula [17], [18]:

$$y^2 \equiv x^3 + ax + b \bmod p \quad (1)$$

$$\text{with } a, b \in GF(p), 4a^3 + 27b^2 \neq 0$$

In this paper, the elliptic curve p-256 in FIPS Pub 186-3 is suggested as an appropriate elliptic curve in the improved protocol [19]. The base point $G = (G_x, G_y)$, the value of a, b, p and the order r of G are given in Federal Information Processing Standards (FIPS) Pub 186-3.

2) Elliptic Curve Diffie-Hellman

Elliptic Curve Diffie-Hellman (ECDH) is an anonymous key agreement protocol that allows two parties, each of which having an elliptic curve public-private key pair, to establish a shared secret over an insecure channel [20]. Assume that two communicating participators A and B possess their own private keys SK_A and SK_B separately. SK_A and SK_B are random integers chosen from the set $\{1, \ldots, r-1\}$. According to private keys, the public keys PK_A and PK_B can be calculated by the base point G of the elliptic curve:

$$PK_A = SK_A \times G,\ PK_B = SK_B \times G \quad (2)$$

Where × is the scalar multiplication of G by an integer. In this key establishment protocol, A and B exchange their public keys and compute (x_k, y_k) by the public key from their opposite side, i.e., A casts $(x_k, y_k) = SK_A \times PK_B$ and B casts $(x_k, y_k) = SK_B \times PK_A$. Therefore, two communicating parties hold the shared key x_k locating at the X coordinate of the point.

B. Security Model

1) Communicating Parties. The improved protocol is executed by the following two parties through an untrusted channel.

- SDF middle device. The SDF middle device lies in the infrastructure layer. It is usually a computationally-limited devices such as a router or a smart phone.
- SDF controller. The SDF controller lies in the control layer. It is much more powerful than the middle device.

2) Attack Model. We specify the attacker has the following abilities.

- The attacker can observe, alter, delay or delete messages transmitted between the coordinator and sensor.

3) Security Goals. Under the above attacker model, security goals of the Unbalanced OpenFunction handshake protocol are listed as follows.

- **Authentication.** After a completed run of the protocol, both the controller and middle device can confirm the received messages are sent by the true communicating parties.
- **Confidentiality.** The session key MK is only known by the controller and middle device in this session.
- **Key freshness.** The session key MK is fresh.
- **Forward secrecy.** When a long-term key is compromised, previous session keys that were established using this long-term key should not be compromised.

IV. Unbalanced OpenFunction protocol

To achieve the improvements for [9] presented in section II, we propose an Unbalanced OpenFunction handshake protocol. This protocol is applied to improve handshake of OpenFunction protocol, which is essential for further updating process.

A. Overview of Protocol

Fig.3 illustrates the processes of setting up a secure connection between OpenFunction device and controller for later transmission of update files. The notations of messages involved in the protocol, which is shown in Fig.3, are explained as follows:

- **SYN**: It is a TCP synchronize message. The SDF middle device initiates a TCP connection by sending the SYN message to the SDF controller.
- **ACK**: It is a TCP acknowledge message. After the SDF controller receives the ACK message from the SDF middle device, the TCP connection is established.
- **OF_HELLO**: It is a message text that is used for version negotiation when a connection is established.
- **OF_FEATURE_REQUEST**: The feature request possessed by the SDF controller is applied for requiring feature of the device.
- **OF_FEATURE_REPLY**: SDF middle device owns feature reply and employs it on feature negotiation. Once the device receives the feature request, the message of feature reply containing the device capability report will be transmitted to the controller.

As shown in Fig.3, the Unbalanced OpenFunction handshake protocol proposed in this paper requires only 4 communications to finish handshake process, which is fewer than 7 communication steps of [9] exhibited in Fig.1.

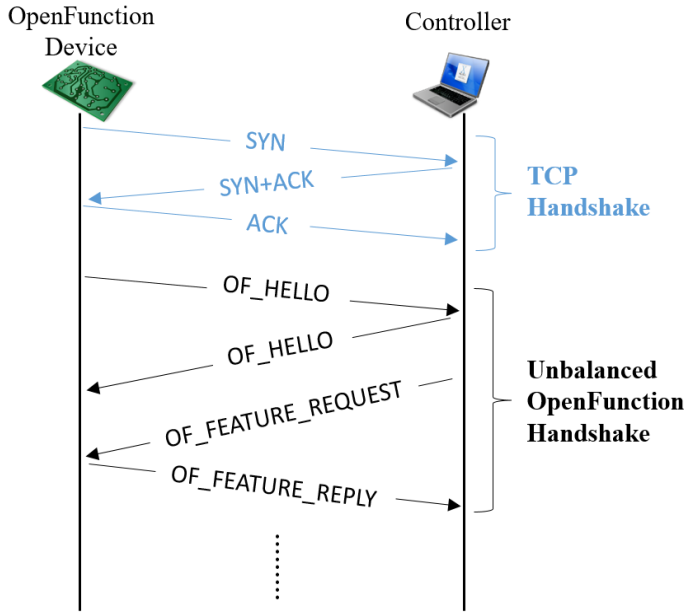


Fig. 3. Process of Unbalanced OpenFunction protocol.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2820086, IEEE Access

B. Protocol Description

Assume that controller and device share their ECC public keys PK with each other. PK_D should be stored previously by controller and PK_C should be held by the device in advance. The ECC private keys SK are only possessed by their owners before initiation. Here we describe the protocol as follows.

1. D chooses a random number R_D and computes U_D:

$$U_D = R_D + SK_D \tag{3}$$

To exchange random number applied in key agreement, D sends the following message M_1 to C.

$$M_1 = < \text{OF\_HELLO}, D, U_D > \tag{4}$$

2. C chooses a random number R_C and computes U_C:

$$U_C = R_C + SK_C \tag{5}$$

Then C generates T_C by

$$T_C = U_C \times G \tag{6}$$

C sends message M_2 to D:

$$M_2 = < \text{OF\_HELLO}, C, T_C > \tag{7}$$

3. C computes provisional ECDH key K_{DH}:

$$K_{DH} = R_C \times (U_D \times G - PK_D) \tag{8}$$

Then C computes MAC_1 as follows:

$$MAC_1 = HAMC_{256}(K_{DH\_x2}, M_1 \,||\, M_2 \,||\, feature\ request) \tag{9}$$

C packs up MAC_1 and *feature request* and sends M_3 to D:

$$M_3 = < feature\ request, MAC_1 > \tag{10}$$

4. D computes its provisional ECDH key K_{DH} by

$$K_{DH} = R_D \times (T_C - PK_C) \tag{11}$$

Then D computes MAC_1:

$$MAC_1 = HAMC_{256}(K_{DH\_x2}, M_1 \,||\, M_2 \,||\, feature\ request) \tag{12}$$

When D receives the message M_3 from C, D verifies whether its MAC_1 is equivalent to C's. If verification succeeds, D will identify C and mark master key MK by $MK = CMAC_{128}(K_{DH\_x1}, C \,||\, D)$. Next, D computes MAC_2:

$$MAC_2 = HAMC_{256}(K_{DH\_x2}, M_1 \,||\, M_2 \,||\, M_3 \,||\, feature\ reply) \tag{13}$$

Then D transmits *feature reply* and MAC_2 packaged on M_4 to C:

$$M_4 = < feature\ reply, MAC_2 > \tag{14}$$

5. C computes MAC_2' according to its ECDH key K_{DH}:

$$MAC_2' = HAMC_{256}(K_{DH\_x2}, M_1 \,||\, M_2 \,||\, M_3 \,||\, feature\ reply) \tag{15}$$

Deduction 1. C calculates $K_{DH} = R_C \times (U_D \times G - PK_D)$

1.	$K_{DH} = R_C \times (U_D \times G - PK_D)$	<known>
2.	$U_D = R_D + SK_D$	<known>
3.	$PK_D = SK_D \times G$	<known>
4.	$U_D \times G = (R_D + SK_D) \times G = R_D \times G + PK_D$	< 2, 3 >
5.	$U_D \times G - PK_D = R_D \times G$	< 4 >
6.	$K_{DH} = R_C \times R_D \times G$	< 1, 5>

Deduction 2. D calculates $K_{DH} = R_D \times (T_C - PK_C)$

1.	$K_{DH} = R_D \times (T_C - PK_C)$	<known>
2.	$T_C = U_C \times G$	<known>
3.	$U_C = R_C + SK_C$	<known>
4.	$PK_C = SK_C \times G$	<known>
5.	$T_C = (R_C + SK_C) \times G \quad = R_C \times G + PK_C$	< 2, 3, 4 >
6.	$T_C - PK_C = R_C \times G$	< 5 >
7.	$K_{DH} = R_D \times R_C \times G = R_C \times R_D \times G$	<1, 6 >

After receiving M_4 from D, C verifies if $MAC_2 = MAC_2'$. If the verification executes successfully, C will identify D and mark master key MK by $MK = CMAC_{128}(K_{DH\_x1}, C \,||\, D)$.

Note that D's K_{DH} computed in step 4 is equivalent to C's K_{DH} generated in step 3, though their calculation procedures vary. The formula derivation for proving equivalence of K_{DH} is described in Deduction 1 and 2.

In the result of deduction, R_C and R_D are the random numbers exchanged by C and D during step 1 and 2 in protocol. The commutative law is applicable to ECC scalar multiplication [21], so K_{DH} of C and D are equivalent. Therefore, the public key algorithm can be accomplished securely based on the design of unbalanced computation.

V. Security Analysis

Section III lists the security goals that our protocol is supposed to achieve. This section presents security analysis of our protocol, and illustrates how the goals are achieved. In this section, we will analyze the security of our protocol according to the security model in Section III.C. We first rewrite the definition of security goals through Proposition 1, 2, 3 and 4. Then, we prove how these propositions stand.

A. Authentication

Proposition 1. After a completed run of the new protocol, D can confirm OF_HELLO, C, T_C and feature_request are sent by C; C can confirm OF_HELLO, D, U_D and feature_reply are sent by D.

Proof (Sketch). First, authentication of *OF_HELLO*, C, T_C and *feature_request* are guaranteed by MAC_1. To compute MAC_1, K_{DH} is required. Since $K_{DH} = R_C \times (U_D \times G - PK_D) = R_D \times (T_C - PK_C)$, the party can calculate K_{DH} only when (R_C, U_D, PK_D) or (R_D, T_C, PK_C) is known. Since R_D and R_C are not transmitted directly in the protocol, purely D and C hold the two values. That is, K_{DH} can only be calculated by D and C; MAC_1 can only be calculated by D and C. Therefore, as long as MAC_1

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2820086, IEEE Access

TABLE 2
EVALUATION OF THE UNBALANCED OPENFUNCTION PROTOCOL'S COST

Communication cost of C	*Communication cost of D*	*Computing cost of C*	*Computing cost of D*
$2\mathcal{M}$	$2\mathcal{M}$	$3\mathcal{S}+\mathcal{R}+3\mathcal{H}$	$\mathcal{S}+\mathcal{R}+3\mathcal{H}$

passes the verification of step 4, *OF_HELLO*, *C*, T_C and *feature_request* are sent by *C*.

Second, authentication of *OF_HELLO*, *D*, U_D and *feature_reply* are guaranteed by MAC_2. The analysis is similar as above.

B. Confidentiality

Proposition 2. After a completed run of the improved protocol, the new shared session key MK is only known by C and D.

Proof (Sketch). As we proved in Proposition 1, K_{DH} can only be calculated by *C* and *D*. Thus, merely *C* and *D* are aware of $MK = CMAC_{128}(K_{DH\_x1}, C \| D)$.

C. Key Freshness

Proposition 3. After a completed run of the improved protocol, both C and D can confirm that MK is fresh.

Proof (Sketch). Both *C* and *D* compute *MK* as $MK = CMAC_{128}(K_{DH\_x1}, C \| D)$. K_{DH} involves two random values R_C and R_D generated by *C* and *D*. Since random values are fresh in each protocol session, both *C* and *D* can confirm that K_{DH} is fresh. Consequently, both *C* and *D* can confirm the freshness of *MK*.

D. Forward Secrecy

Proposition 4. Assume the attacker compromises long term secret keys SK_D and SK_C. The attacker is unable to compromise previous session keys established using SK_D and SK_C.

Proof (Sketch). Since $MK = CMAC_{128}(K_{DH_x1}, C \| D)$and $K_{DH} = R_C \times (U_D \times G - PK_D) = R_D \times (T_C - PK_C)$, computing the previous session key requires the random values R_C or R_D in that session. Only with SK_D and SK_C, the adversary is unable to compute K_{DH} in previous sessions. Hence, it can be confirmed that the improved protocol provides forward secrecy.

VI. PERFORMANCE

To observe the performance of the proposed protocol, we evaluate the computation and communication cost theoretically. In addition, a set of experiments are carried out to test protocol's performance.

A. Theoretical Evaluation

Denote the message transmitted in communication by $\mathcal{M}$, the algorithm HMAC by $\mathcal{H}$, the operation of ECC scalar multiplication by $\mathcal{S}$, the operation of ECC point subtraction by $\mathcal{R}$. In Table 2, we have evaluated the communication and computation cost of both sides, i.e. *C* and *D*, on the recommended protocol.

Based on Table 2, we are able to draw following conclusions.

- Communication costs of the unbalanced OpenFunction protocol are acceptable. The total communication costs are $4\mathcal{M}$ and the number of messages transmitted by the middle device is equal to the controller. Practically the communication on *C* might be more lightweight than that on *D* while the controller is more powerful than the device.
- Computing costs of the improved protocol are acceptable. The total computing cost is $4\mathcal{S} + 2\mathcal{R} + 6\mathcal{H}$, but the computation of two sides is asymmetric. We can conclude from the data that the computational load on the device is lower than that on the controller because the computing cost of the device is less than the controller by $2\mathcal{S}$. The performance is adequate for capability-limited devices.

B. Experiment

To verify the above results, we design several experiments deploying the protocol suite to test the performance. The controller is allocated on a laptop and the middle device is

TABLE 4
AVERAGE RUNTIME (IN SECOND) OF COMPUTING IN THE PROTOCOL FOR DIFFERENT CURVES

Curve	*Computing time on C*	*Computing time on D*	*Total computing time*
P-192	0.046000004s	0.004415756s	0.050415760s
P-224	0.061400008s	0.004478021s	0.065878029s
P-256	0.080500031s	0.017669898s	0.098169929s
P-384	0.171099925s	0.020273378s	0.191373303s

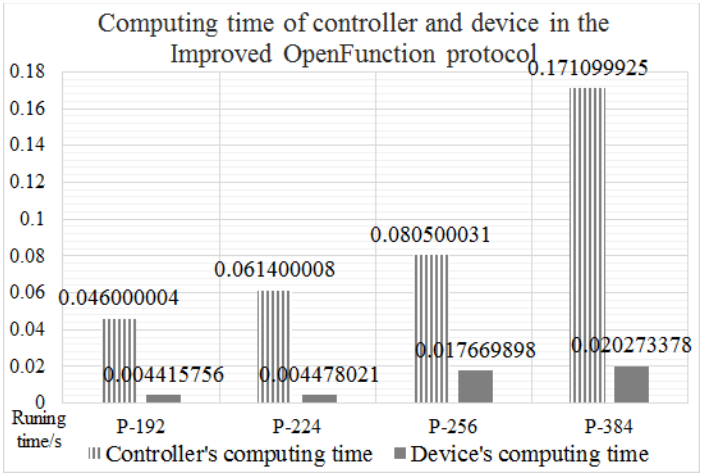


Fig. 4. Average runtime (in second) of computing in the protocol for different curves. The protocol requires an asymmetrical computational load on *C* and *D*.

TABLE 3
DETAILS ABOUT THE EXPERIMENT DEVICES

Role	*Experiment*	*Detail*
Controller	Laptop	CPU: 2.50GHz(i7-4710MQ), Memory: 8GB, Solid State Disk: 256GB
Device	Raspberry Pi3	CPU: 1.2GMHz 64-bit ARM v8, Memory: 1GB, Storage: 32GB

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2820086, IEEE Access

TABLE 5
AVERAGE RUNTIME (IN SECOND) OF COMMUNICATION IN THE PROTOCOL FOR DIFFERENT CURVES

Curve	*Total communication time*
P-192	0.001259894s
P-224	0.001343411s
P-256	0.001814200s
P-384	0.002674742s

configured on a Raspberry Pi. More details of experiment environment are illuminated in Table 3. The protocol suite realizes the unbalanced OpenFunction protocol. HMAC is implemented with the Secure Hash Algorithm (SHA) 256. According to the recommendation of FIPS Pub 186-3 [19], we implement the ECDH algorithm by several suggested curves, i.e. Curve P-192, P-224, P-256, and P-384.

We ran the protocol suite with each curve for 10 times. The average runtime of computing in each curve's experiments is shown in Table 4 and Fig.4. Additionally, all data about the average runtime of communication is illustrated in Table 5. The value in Table 4 and Table 5 presents that the average runtime of *C* and *D* increases as the bit length of ECC generated key increases. The length of ECC key is relative to its elliptic curve, i.e. each element of generated key pair on Curve P-192 retains the length of 192 bit. Fig.4 reveals that the difference in performance between controller and device is prominent. In all cases of Fig.4, the computational burden on the Raspberry Pi is lower than that on the laptop. It is known that the computing power of the laptop is much higher than the Raspberry Pi, so the modified protocol substantially reduces computing burden on *D*.

VII. RELATED WORK

In this section, we provide a brief overview of some related research work; and do theoretical or experimental comparison between them and the unbalanced OpenFunction protocol.

A. Overview of Related Work

There are two previous generations of the Unbalanced OpenFunction handshake protocol sketchily introduced in the following.

Xue et. al [9] have introduced an intelligent building framework called S^2Net, which is based on Software Defined Networking (SDN) architecture. The two protocols of S^2Net are the origin of OpenFunction protocol. The first is to initialize connection, authenticate communicating parties and negotiate session key. The second is to transmit messages encrypted and decrypted by session key. The process of first protocol is descried in Fig.2.

In [8], Xue et. al propose a new architecture named Software Defined Function (SDF) derived from SDN. Instead of OpenFlow only providing packet forwarding across switches in SDN, OpenFunction protocol is presented for remote updating of IoT devices. As the improvement of S^2Net, OpenFunction finishes authenticated handshake within 4 steps in first part and then messages updating files in second part. However, [8] only advances conception and overview of OpenFunction except protocol description and detail.

Furthermore, SSL and TLS are traditional standard protocols aimed to ensure security of HTTP and other Internet applications. As shown in Table 7, SSL and TLS are widely applied for remote system updating of common operating systems, such as Windows, Debian, iOS and Android.

B. Comparison

The comparison in accordance with features of the proposed protocol and its predecessors, i.e. [8], [9], is listed in Table 6. Differing with these two protocols, based on ECC algorithm, the unbalanced OpenFunction protocol can directly complete authentication without the secret key before beginning. Therefore, it does not require the two sides to exchange and store secret keys by specific channels or clipper chips. While the other two protocols spend balanced computational cost on both sides, the protocol recommended in this paper adopts unbalanced computation to reduce the burden on the devices with limited computing capacity.

In addition, in terms of authentication, we compare this protocol with the remote system update or Device Firmware Update (DFU) of many current schemes in several kinds of mobile operating systems, such as Windows and Debian in PCs, as well as Android and iOS in mobile phones, and conclude the comparative results in Table 7. According to the hash algorithms listed in Table 7, we can recognize that SHA-1 is adopted in digital signature or key fingerprint of most of the remote system upgrade, and yet the Debian system and the proposed protocol utilize a famous member of the SHA-2 family, SHA-256. However, the SHA-1 algorithm was officially deprecated by NIST in 2011 [22] because of its dubious security. Moreover, SHA-1 is suggested to be replaced with SHA-2 algorithm family by SSL Labs [23]. Thus, the defensiveness of the improved protocol is perhaps higher than those comparative protocols. According to Wander's statement [24], the benefits of ECC over RSA are less computation, less transmitted data and shorter secret key in same encryption

TABLE 6
FEATURE COMPARISON OF PROPOSED PROTOCOL AND ITS PREDECESSORS
("√" DENOTES THE PROTOCOL POSSESSES THE FEATURE, AND "×" DENOTES THE PROTOCOL DOES NOT OWN THIS FEATURE)

Feature	***Unbalanced OpenFunction protocol***	***Protocol One of S^2Net [9]***	***OpenFunction protocol [8]***
Employing ECC algorithm in authentication	√	×	×
Not requiring shared secret key in advance	√	×	×
Shifting partial computational load from *D* to *C* (Unbalanced computing cost in *D* and *C*)	√	×	×
Number of handshake steps	4	7	4
Protocol description and detail	√	√	×

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2820086, IEEE Access

TABLE 7
COMPARISON OF AUTHENTICATION IN REMOTE SYSTEM UPDATE OR DEVICE FIRMWARE UPDATE OF FUNCTION STATIONS OPERATED ON SEVERAL MAINSTREAM MOBILE OPERATING SYSTEMS AND REMOTE REPROGRAMMING OF THE PROPOSED PROTOCOL

Operating System	*Protocol of Remote System Update (RSU) / Device Firmware Update (DFU)*	*Handshake Authentication Protocol of RSU / DFU*	*Encryption algorithm of authentication*	*Hash Algorithm*	*Unbalanced cost on server and client in handshake process*
Windows	MS-WUSP (SOAP-based) [3]	SChannel SSP protocol (TLS-SSL protocol) [3], [25]	RSA [25]	SHA-1 [25]	×
Debian	HTTP-based protocol [4]	SSL protocol[4]	RSA [26]	SHA-1 [27]	×
Android	OTA protocol [5]	TLS protocol [31], [32] *	RSA [28]	SHA-1 [5]	×
iOS	OTA protocol [6]	SSL protocol [29]	RSA [29], [30]	SHA-1 [30]	×
\	OpenFunction protocol [9]	Proposed protocol	ECC	SHA-256	√

* Note that as described in [31], OAuth 2.0 is used as authentication protocol for Android Over-The-Air API. In page 16-17 of OAuth 2.0 RFC standard [32], OAuth 2.0 is employed in authentication between client and third-part authentication server, and it applies TLS as authentication protocol between client and resource server. Since we focus on handshake authentication of client and resource server, we choose TLS protocol to make comparisons.

strength so that energy and communication cost can be saved significantly. **Therefore, the proposed protocol authenticated by ECC is more qualified than any other approaches encrypted by RSA to the power-limited and memory-bound devices that act as middle devices.** The unbalanced computational cost of the protocol is also a specific advantage in diminishing devices' burden.

Moreover, we also implement the secure handshake process of updating schemes used by Windows, Debian, iOS and Android, i.e. SSL handshake and TLS handshake, and compare their performance with our protocol. In our proposed protocol, we assume that *C* and *D* have exchanged their public key before OpenFunction handshake. Therefore, in order to test data more veritably and equitably, we eliminate the time of producing and exchanging Certificate Authority (CA) and employ ECDH algorithm to achieve ECC encryption in the experiments of SSL 3.0 protocol and TLS 1.2 protocol. In the experiment, we test the runtime of SSL on *C* and *D* in the same experimental environment listed in Table 3. We also run SSL handshake and TLS handshake with ECDH algorithm on curves P-256 and P-384 for 10 times; and the results are presented in Table 8. Furthermore, the average computing runtime of device in SSL handshake, TLS handshake and the unbalanced OpenFunction protocol are compared in Fig.5. From this figure, we can conclude that the proposed protocol enables devices to run much faster than devices executing SSL handshake or TLS handshake. In this fair situation, the total controller runtime of P-256 and P-384 in the unbalanced OpenFunction protocol are about 0.0982s and 0.1914s respectively, which are less than 0.2406s and 0.2650s in SSL handshake and also shorter than 0.2455s and 0.2687s in TLS handshake. **Therefore, the unbalanced OpenFunction protocol can practically reduce computing cost of devices and achieve better efficiency of handshake than SSL and TLS.** In addition, observations of Fig.5 suggest that the runtime of TLS is slightly longer than SSL in both curves. Therefore, SSL is more suitable to pursue greater efficiency. As the differences between SSL and TLS explained by Thomas [33], since TLS can operate standardized MAC (HMAC) with not only MD5 and SHA but also any other hash function, TLS provides more flexible selections to developers.

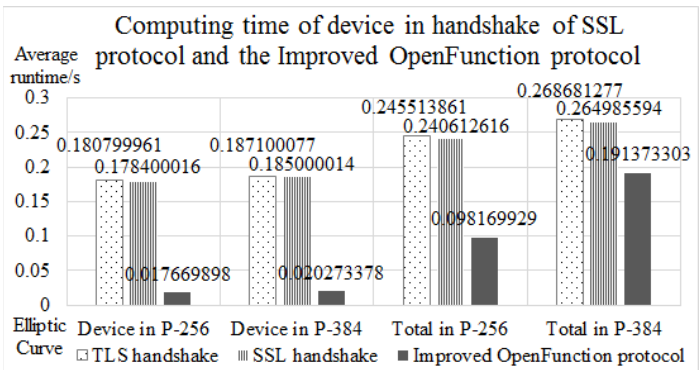


Fig. 5. Average runtime (in second) of device's and total computing time in SSL handshake and the unbalanced OpenFunction protocol.

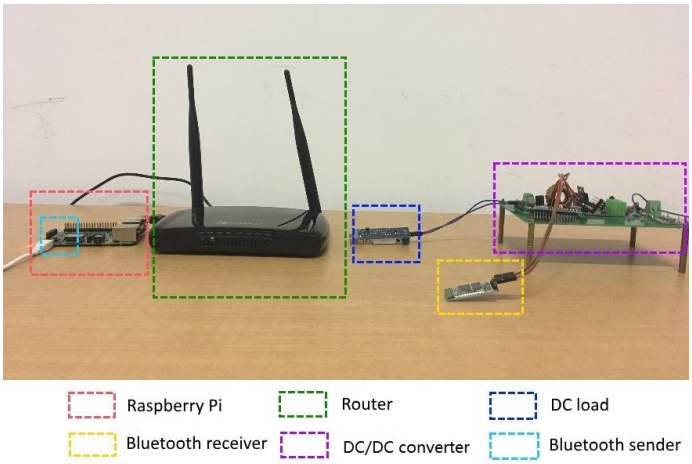
VIII. USE CASE

As in the previous description, our unbalanced OpenFunction protocol has a significant strength on lessening the computational load of function stations during wireless reprogramming, which is advanced in some applications of wireless sensor network. In this section, we illuminate the usage improved by our protocol through a use case named SDF-based photovoltaic (PV) energy system. Traditional schemes apply SSL or TLS in building reliable connection for transmission, the middle device in this PV system is less powerful and may spend too much cost to reach the standard runtime in industry. Furthermore, many actual solar energy companies have high requirements in timeliness in data transmission of PV panels. However, slow or even overburdened middle device might cause the consequence that statistical accuracy of data becomes beyond reasonable range. Therefore, this use case employs the improved protocol into SDF-based PV energy system to reduce computing cost in middle device and improve efficiency of updating. The protocol updates Direct Current/Direct Current

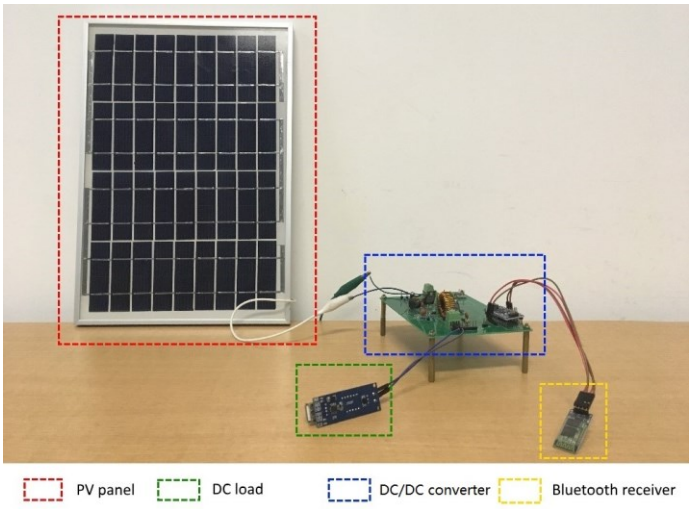
TABLE 8
AVERAGE RUNTIME (IN SECOND) OF COMMUNICATION IN SSL HANDSHAKE AND TLS HANDSHAKE PROCESS

Protocol	*Curve*	*Computing time on C*	*Computing time on D*	*Total computing time*
SSL	P-256	0.0622126s	0.178400016s	0.240612616s
	P-384	0.0799856s	0.185000014s	0.264985594s
TLS	P-256	0.0647139s	0.180799961s	0.245513861s
	P-384	0.0815812s	0.187100077s	0.268681277s
Proposed protocol	P-256	0.080500031s	0.017669898s	0.098169929s
	P-384	0.171099925s	0.020273378s	0.191373303s

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2018.2820086, IEEE Access



(a) OpenFunction reprogramming system



(b) PV system

Fig. 6. SDF-based PV energy system.

(DC/DC) converter algorithm from remote servers to the DC/DC converter. In addition, the ECC algorithm used in the improved protocol requires two sides to store shorter secret key, which has the advantage in memory-bound devices like DC/DC converter.

According to control algorithm in DC/DC converter, the direct current generated by PV panels can be converted from a higher input voltage to an adequate output voltage that can be accepted by a variety of middle devices. By utilizing the improved protocols of SDF, the PV energy systems can securely upgrade the DC/DC converter algorithm through wireless links. The SDF-based PV panel system involves the following three phases, and our improved protocol is included in the first step. The demo system is realized as shown in Fig.6, and the details about the experiment devices are itemized in Table 9.

1. **Establishing communication**. The remote server (such as a laptop) and the middle device (such as a Raspberry Pi) execute the Unbalanced OpenFunction handshake protocol to build a communication channel. As a result, a relative master key is shared between each other.
2. **Transmitting update files.** To transmit update files from the server to the middle device, these two devices execute the extended OpenFunction messaging protocol of SDF so that the middle device can receive new algorithm files.
3. **Reprogramming end devices.** The middle device reprograms the end devices (i.e. the DC/DC converter) by the update files.

Note that the DC load consumes energy and can be used to measure the output voltage, and the Buck converter is recognized as a DC/DC converter in the demo system. Although the Raspberry Pi, as a middle device, is a less capable device compared with a normal laptop, the Unbalanced OpenFunction handshake protocol in the first step reduces the computational cost of the middle device. The unbalanced computing cost strategy enhances the efficiency of the SDF-based PV energy system.

IX. Conclusion

At present, wireless updating has been widely and frequently used in IoT applications for system upgrade or device firmware update. In addition to SSL protocol and TSL protocol which are applied for most mobile devices, OpenFunction protocol based on SDF can support IoT devices to build secure communications with servers and accept new system files. After that, these devices can execute the system files to update their sensors by some WSN reprogramming methods. In OpenFunction protocol, handshake process is to guarantee communication security of wireless updating. The OpenFunction handshake requires equal computation cost on both sides. However, in fact, computing powers of devices and controllers are different. The OpenFunction protocol ignores the advantage that servers possessing high computing capability can share computational work of devices so as to reduce devices' burden. Therefore, by making use of this advantage, this paper has proposed an unbalanced OpenFunction protocol to decrease devices' computational load and improve efficiency of OpenFunction handshake. In comparison with original protocol, the unbalanced OpenFunction protocol completes handshake process with fewer steps and distribute less load on device by allocating unbalanced computation tasks in two sides. The proposed protocol also demands no secret key as precondition and firstly secures authentication with ECC algorithm.

TABLE 9
Details about the devices in SDF-based PV energy system

Component	*Detail*
Raspberry Pi	CPU: 1.2GMHz 64-bit ARM v8, Memory: 1GB, Storage: 32GB
DC load	Input voltage: 4.5V-40V
PV panel	10W, Maximum power voltage: 17.5V
Laptop	CPU: 2.50GHz (i7-4710MQ), Memory: 8GB, Solid State Disk: 256GB
Bluetooth sender	BCM3438 Wi-Fi/BLE chip
DC/DC converter	Maximum Input voltage: 25V, Output voltage: 0V-25V

Moreover, after implementing and testing in same experimental environment, we concluded that the unbalanced OpenFunction protocol has better performance than SSL handshake and TLS handshake, and it assuredly reduces much computing cost of device. Thus, the proposed protocol is particularly suitable for not only the use case exemplified in Section 7 but also IoT access control applications, such as smart lock systems or intelligent door security systems, to efficiently secure their wireless communication for system update.

In the future, we plan to carry out experiments of whole wireless updating process including message transmission, and test the performance with different ECC curves. Moreover, we will deploy our protocol in large scale WSNs to test its performance.

REFERENCES

[1] Atzori L, Iera A, Morabito G. The internet of things: A survey [J]. Computer networks, 2010, 54(15): 2787-2805.

[2] Rose K, Eldridge S, Chapin L. The internet of things: An overview [J]. The Internet Society (ISOC), 2015: 1-50.

[3] "[MS-WUSP]: Windows Update Services: Client-Server Protocol Specification", Microsoft Corporation, Dec. 14, 2011; pp. 6-7, 9-10, 102-103.

[4] "Release Notes for Debian 8 (jesssie), 64-bit PC", Free Software Foundation, Inc., May 14, 2016; pp. 9-22.

[5] "Inside OTA Packages". (2017, Mar. 27), *OTA Updates (Android)*, Google Corporation, Retrieved Aug. 5, 2017. [Online]. Available: https://source.android.com/devices/tech/ota/inside_packages

[6] "IOS Security – White Paper", Apple Corporation, Sep. 2015; pp. 5-6.

[7] Zhou L, Wu D, Zheng B, et al. Joint physical-application layer security for wireless multimedia delivery [J]. IEEE Communications Magazine, 2014, 52(3): 66-72.

[8] Xue N, Liang L, Zhang J, et al. POSTER: A Framework for IoT Reprogramming[C]//International Conference on Security and Privacy in Communication Systems. Springer, Cham, 2016: 751-754.

[9] Xue N, Huang X, Zhang J. S2Net: A Security Framework for Software Defined Intelligent Building Networks[C]//Trustcom/BigDataSE/ISPA, 2016 IEEE. IEEE, 2016: 654-661.

[10] Xue R, Huang X, Zhang J, et al. Software defined intelligent building [J]. 2015.

[11] Sood K, Yu S, Xiang Y. Software-defined wireless networking opportunities and challenges for internet-of-things: A review [J]. IEEE Internet of Things Journal, 2016, 3(4): 453-463.

[12] Han D M, Lim J H. Design and implementation of smart home energy management systems based on zigbee [J]. IEEE Transactions on Consumer Electronics, 2010, 56(3).

[13] Carlini J. The Intelligent Building Definition Handbook [J]. IBI, Washington, DC, 1988.

[14] Philips. Philips, *2015 annual report*. [Online]. Available: http://www:philips:com/corporate/resources/annual-results/2015/PhilipsFullAnnualReport2015English:pdf, 2016.

[15] Zhang J, Xue N, Huang X. A Secure System For Pervasive Social Network-Based Healthcare [J]. IEEE Access, 2016, 4: 9239-9250.

[16] Diffie W, Hellman M. New directions in cryptography [J]. IEEE transactions on Information Theory, 1976, 22(6): 644-654.

[17] Koblitz N. Elliptic curve cryptosystems [J]. Mathematics of computation, 1987, 48(177): 203-209.

[18] Miller V S. Use of elliptic curves in cryptography [C]//Conference on the Theory and Application of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1985: 417-426.

[19] Locke G, Gallagher P. Fips pub 186-3: Digital signature standard (dss) [J]. Federal Information Processing Standards Publication, 2009, 3: 186-3.

[20] Barker E, Chen L, Roginsky A, et al. Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography [J]. NIST special publication, 2013, 800: 56A.

[21] Longa P. Accelerating the scalar multiplication on elliptic curve cryptosystems over prime fields [D]. University of Ottawa (Canada), 2007.

[22] Barker E, Roginsky A. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths [J]. NIST Special Publication, 2011, 800: 131A.

[23] Ristić, I. (2012). SSL/TLS deployment best practices. vol. 1.3, 17 September 2013, 12.

[24] Wander A. S, Gura N, Eberle H, et al. Energy analysis of public-key cryptography for wireless sensor networks [C]//Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on. IEEE, 2005: 324-328.

[25] Microsoft Corporation, "TLS/SSL Settings". (2017, Sep. 20), Retrieved Oct. 11, 2017. [Online]. Available: https://technet.microsoft.com/en-us/library/dn786418(v=ws.11).aspx

[26] "Self-Signed Certificate". (2014, Apr. 27), *Wiki of Debian*, Free Software Foundation, Inc., Retrieved Aug. 5, 2017. [Online]. Available: https://wiki.debian.org/Self-Signed_Certificate

[27] "SSLkeys". (2011, Apr. 25), Wiki of Debian, Free Software Foundation, Inc., Retrieved Aug. 5, 2017. [Online]. Available: https://wiki.debian.org/SSLkeys

[28] "Signing Builds for Release". (2017, Jun. 20), *OTA Updates (Android)*, Google Corporation, Retrieved Aug. 5, 2017. [Online]. Available: https://source.android.com/devices/tech/ota/sign_builds

[29] "Creating a Profile Server for Over-The-Air Enrollment and Configuration". (2014, Apr. 17), *Over-the-Air Profile Delivery and Configuration*, Apple Corporation, Retrieved Aug. 5, 2017. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/iPhoneOTAConfiguration/profile-service/profile-service.html#//apple_ref/doc/uid/TP40009505-CH2-SW2

[30] "iOS 7: List of available trusted root certificates". (2017, Feb. 24), *Apple Support*, Apple Corporation, Retrieved Aug. 5, 2017. [Online]. Available: https://support.apple.com/en-us/HT203065

[31] "Authorize Requests". (2017, Oct. 16), *Android Over The*

Air API, Google Corporation, Retrieved Nov. 1 2017. [Online]. Available: https://developers.google.com/android/over-the-air/v1/how-tos/authorizing

[32] Hardt D. "RFC6749 - The OAuth 2.0 Authorization Framework". Oct. 2012[J]. Retrieved Nov. 1 2017. [Online]. Available: https://tools.ietf.org/html/rfc6749

[33] Thomas S. SSL and TLS essentials [J]. New Yourk, 2000: 3.