





Self-secure Communication for Internet of Things

Bin Hao^{1,2}  and Sheng Xiao^{1,2} 

¹ College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
{binhao, xiaosheng}@hnu.edu.cn

² National 2011 Collaborative Innovation Center for High Performance Computing, Changsha, China

Abstract. Cryptographic key management is a challenge for the large scale deployment of Internet of Things (IoT) devices. It is difficult to properly setup and constantly update keys for numerous IoT devices, especially when these devices are restricted by size and lack of the key input interface. This paper proposes a lightweight key management scheme which embeds the key distribution and update process into the communication process. The keys are constantly changing as the communication data flowing back and forth between IoT devices. Therefore even if a key is stolen by the attacker, it will quickly become invalid as the communication goes on. The proposed scheme also contains a key initialization protocol which generates independent keys for multiple IoT devices simultaneously. This paper describes the protocols in detail and analyzes its security properties. The practicality of the protocol is verified by experiments.

Keywords: Wireless randomness · Key agreement · Secure communication · Internet of Things

1 Introduction

The Internet of Things (IoT) technology allows the ordinary objects that perform independent functions to achieve interconnection [1]. With the communication ability, IoT devices provide great convenience to people's lives such as to create a smart home, to help monitoring the patients, and to allow a vehicle to sense its surroundings [2].

Since the communication among IoT devices carries the sensitive data which closely associates with the people's private lives, it becomes a prominent target for the malicious attackers [3]. Moreover, IoT devices are often restricted by the form factor and the power supply, the Internet security solutions are not entirely suitable for the IoT environment [4]. One particular challenge for the IoT communication security is the key management. It is difficult to preset cryptographic keys in the IoT devices as they are great in numbers and the IoT device manufacturers have no information about when these devices would be installed and what would be the network topology. It is even more difficult to update the keys in IoT devices as they are often scattered in the fields. It would be desirable

to have a key management scheme that allows the IoT devices to negotiate and update the keys by themselves, in the field, with negligible performance penalty. The paper responds to the challenge and proposes a key management scheme that enables self-secure communication for Internet of Things.

Self-secure communication does not rely on the PKI infrastructure or any pre-shared key material. It collects the physical layer randomness in wireless transmissions and converts the randomness into symmetric keys. Unlike traditional physical layer security methods such as [5–8], our scheme does not go deep into the physical channel status or coding schemes. Our scheme embeds the key management in the communication process and takes advantages of the error re-transmission mechanism, therefore minimizes the security overhead on communication bandwidth or delay. The main idea of self-secure communication is to utilize the inevitable packet loss phenomenon in wireless communications. Particularly for the IoT devices, because the transmission power limit, such packet loss phenomenon occurs for both the benign user IoT devices and for any attacker who attempts to eavesdrop the IoT communication.

The main contributions of this paper are:

1. This paper proposes a lightweight key management scheme which allows IoT devices to autonomously establish and update keys without pre-shared key materials or pre-installed public key certificates.
2. The proposed key management scheme supports the secure one-to-many communications and the secure relay communications. These communication modes are essential for IoT applications.
3. The proposed key management scheme considers the abnormal situations such as node failures, power outages, and connection losses. It could be easily implemented into engineering solutions.
4. This paper verified the practicality of the proposed scheme using experiments.

The rest of the paper is organized as follows: Sect. 2 briefs the previous works related to our research. Section 3 describes the proposed self-secure communication scheme. Section 4 verifies the feasibility and robustness of our protocol by experiments. Section 5 summarizes the paper.

2 Related Work

Since the concept of “Internet of Things” was proposed, researchers have been searching for effective key management protocols with the constraint of the limited resources of IoT nodes. The previous works could be roughly divided into three categories: trusted third parties, proxy-based encryption calculations, and batch processing. The key management protocol proposed in [9–11] is based on the public key system of a trusted third party and requires a trusted certification authority to issue a certificate for it or generate and distribute keys. J. Shen et al. [12–14] describe agent-based end-to-end key management protocols. The core idea of these three protocols is to delegate the complex key calculation operations in the public key encryption system to multiple adjacent unrestricted or less-constrained nodes for execution. These nodes are called agents. Each agent participates

in the calculation and transfer of a sender's public DH (Diffie-helloman) key, and also participates in the derivation of the public DH key before the sender is constructed. This approach reduces the computational pressure on the communication nodes. References [15, 16] proposed a group key management protocol based on batch processing. The group key is distributed securely to all group members through a key distribution center (KDC). If a node joins or leaves, the KDC will establish the new group key and send it to all group members again.

The above-mentioned protocols are basically aimed at the problem of limited node resources in the Internet of Things, focusing on how to design lightweight key management protocols. However, the Internet of Things has some characteristics that are different from the traditional Internet. First, the nodes cannot be specially monitored and inspected [17, 18]. Some nodes in the IoT system are deployed in some complex geographical environment [20, 21], and it is difficult to achieve real-time monitoring and inspection of each node by a dedicated administrator. This may cause the keys to be lost for a long time without being discovered, leading to a leak. Second, some scenarios lack human-computer interaction interface [22–24]. Some terminal nodes of the IoT system lack the means to interact with key managers, and many devices still need to manually enter the keys regularly to update them. For example, if the smart home requires workers to come to the home to update the keys regularly, and leave an interface for external devices to connect, this is neither convenient nor secure.

3 Self-secure Communication

The main challenge to secure IoT communication is that the IoT devices need to operate in a fully autonomous manner. Therefore, it is necessary to have the IoT devices to negotiate and establish the initial keys *after* the physical deployment of the IoT network and frequently update keys by themselves. Some may argue that to have a public key infrastructure (PKI) with a unified root certificate authority (CA) would help solve the key management problem. However, to have PKI reachable for all IoT devices is over demanding in many application scenarios. To have an interoperable root CA mechanism is even more impractical since IoT devices could come from many independent vendors around the world. In this paper, we propose the self-secure communication scheme that allows IoT devices to autonomously establish and update keys, without the need of pre-shared key materials or public key certificates. Moreover, the self-secure communication is resistant to the key theft attacks. The attacker may obtain the key and compromise the communication security for a short period of time. The communication security could automatically recover as the communication goes on. All these security features are based on the inevitable, random packet losses in the wireless communications among IoT devices.

The self-secure communication involves two phases: key establishment and key updates. In both phases, only symmetric key cryptography is needed. Without loss of generality, this paper uses Alice-Bob-Eve model to illustrate the self-secure communication protocols. As shown in Fig. 1, the attacker Eve is allowed to eavesdropping and injecting the communication between Alice and Bob.

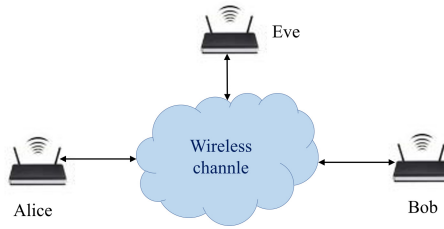


Fig. 1. Secure communication with the eavesdropping attacker

3.1 Notations

Table 1 summarizes the symbols used for the rest of the paper.

Table 1. Point to multipoint key negotiation situation

Symbol	Definition
P_p	The constructed plaintext packet
P_m	The constructed temporary packet
P_c	The constructed ciphertext packet
N	Minimum number of random messages sent by the sender to negotiate the key
M	Minimum number of random messages the receiver uses to negotiate the key
K_0	Initial communication key
K_D	Derived key
K_i	Communication encryption key
α	Communication times parameter
W	Number of communication rounds where the sender stores the key to the hard disk during the communication phase
K_{j*W}	The jth W round key
Ω	Maximum number of retransmissions of the sender during the communication phase

Figure 2 shows the packet format used in the self-secure communication protocols. A data packet contains the following fields: phase identification (Phase), sequence number (Seq), retransmission information (Retran), message length (Len), random data (Rand), communication message (Message) and message authentication code (HMAC), as shown in Fig. 2.

Phase	Retran	Len	Seq	Rand	Message	HMAC
-------	--------	-----	-----	------	---------	------

Fig. 2. Packet format

3.2 Key Establishment Stage

It is assumed that Alice and Bob are two IoT devices without any pre-shared secret information, nor do they have any public key certificates installed. Figure 3 illustrates the key establishment process.

- The sender Alice sends a request negotiation packet

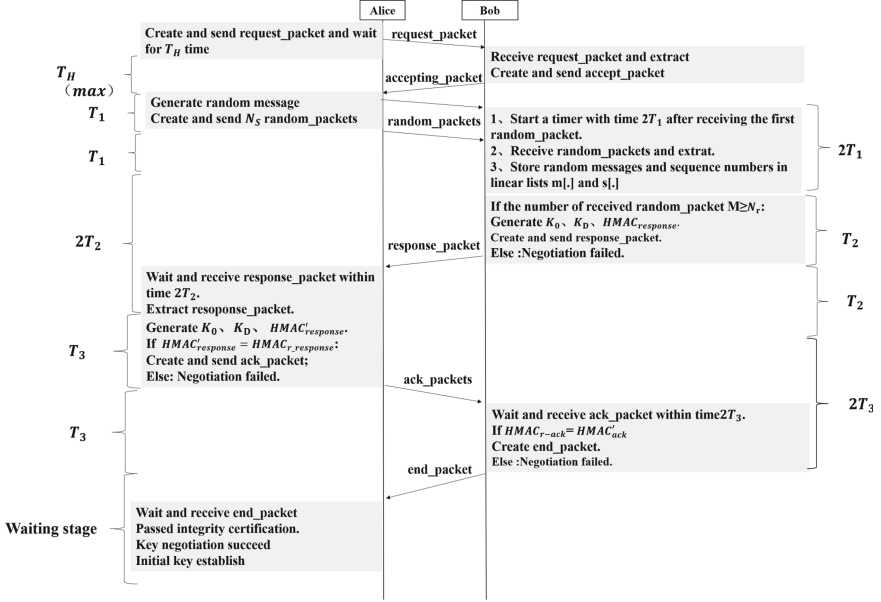


Fig. 3. Key negotiation process

Alice wants to communicate with Bob, so construct a request packet named `req_pkt`, and fills in the value of each field according to the format of the packet specified by the protocol. At this time, the value of Phase(P) is set to 0; the Retran(R) field is set to 0; The Message(M) field is set to Alice's identification information, such as the MAC address; the Len(L) field is set according to the length of the value of the Message field; the Seq(S) field is set from 0.

Then concatenate these fields to get the Plaintext(P_p):

$$P_p = (P \| S \| T \| L \| R) \quad (1)$$

Next, the P_p is filled by OAEP function to get a temporary data packet Mediantext(P_m), to prevent short message attacks. As shown in formula (2):

$$P_m = f_{OAEP}(P_p, a) \quad (2)$$

a is a one-time random number generated by Alice.

Then calculate the hash value $Hash_A$ according to formula (3).

$$Hash_A = f_{Hash}(P_m) \quad (3)$$

The req_pkt is constructed by connecting P_m and Hash_A .

Finally, Alice sends it to the receiver and starts a wait timer with the time set to T_H .

– Receiver sends rcv_pkt

Bob receives and extracts the req_pkt to get $\text{Mediantext}_r(P_{m \cdot r})$ and Hash_r . Then the plaintext data packet $P_p \cdot r$ are calculated by the OAEP solution function whose parameter is $P_m \cdot r$. and the Hash_r^l is calculated according to formula (5), which is used to compare with the received HMAC_r to verify the completeness.

$$P_{p \cdot r} = f_{OAEP}^{-1}(P_{m \cdot r}) \quad (4)$$

$$\text{Hash}_r^l = f_{Hash}(P_{m \cdot r}) \quad (5)$$

Bob determines whether the equation $\text{Hash}_r^l = \text{Hash}_r$ is true. If it is true, he will extract the value of the Message field and verify that it conforms to the MAC address format. If so, Bob will consider whether to agree to negotiate a key with Alice. When Bob is unwilling to communication with Alice, he will remain silent. If Bob frequently receives Alice's request packets later, he will reject each packet within T_d time. If Bob also wants to negotiate a key with Alice, he will construct an accept packet named accept_pkt, set the value of the Message field to Bob's identification information, such as the MAC address, and send it to Alice.

– The sender sends a random message to negotiate the key.

If Alice does not receive the accept_pkt sent by Bob within T_H time, she will wait for T_W time before sending the next request. If Alice receives the accept_pkt and passes verification, she will send random data packets as the key negotiation material. Alice sends N_S random message packets rand_pkt_{*i*} ($i = 0, 1, 2, \dots, N_S$) within T_1 time. The Phase field of these packets is set to 0; the value of the Seq field are set from 0 to N_S in sequence according to the sending order, which are represented by Seq_{*i*} ($i = 0, 1, 2, \dots, N_S$); the value of the Message field are set by information randomly generated by Alice, expressed in Randmsg_{*i*} ($i = 0, 1, 2, \dots, N_S$). After sending all the data packets, a timer T_1 is started to wait for Bob's reply, so that Bob has enough time to receive the data packets. The sender saves all random messages that have been sent and the corresponding sequence numbers in memory and hard disk.

– Receiver sends serial number set and generates initial key

After receiving the first rand_pkt, Bob starts a timer with a time of $2T_1$. During this period, each time Bob receives a rand_pkt, he extracts the content of each field and performs integrity and sequence number check. During the period of verification, if there are two packets that have the same sequence number, the two packets will be directly discarded and the sequence number is added to the blacklist; If the random message data packet passes the integrity and the sequence number check, the values of Message field and Seq field will be stored in the memory. Bob keeps two linear lists, m[.] and s[.], m[.] is used to store the value of the Message field, and s[.] is used to store the value of the Seq field in the same order as m[.].

When the $2T_1$ timer expires, Bob starts a $2T_2$ timer and checks whether the number of received packets M exceeds a preset threshold N_r . If $M \geq N_r$ is not true, Bob reports to the upper layer: the negotiation fails and the initial key cannot be generated; If $M \geq N_r$ holds, Bob generates an initial key K_0 . Assume that all completely received

random data is represented as $m[i]$ ($i = 0, 1, 2, \dots, M$), Bob uses the hash function, such as $\text{sha256}(\cdot)$, to process $m[i]$ to get the message digest $m_h[i]$ ($i = 0, 1, 2, \dots, M$). As shown in formula (6):

$$m_h[i] = f_{Hash}(m[i]) \quad (6)$$

After obtaining the list $m_h[i]$, Bob calculates the Key_0 .

$$K_0 = f_{Hash}(m_h[0] + \dots + m_h[i] + \dots + m_h[M]) \quad (7)$$

In order to ensure the security of the initial key K_0 , the derived function is used to encrypt the information to be transmitted, he uses the key derivation function $\text{kdf}(\cdot)$ to generate the derived key K_D :

$$K_D = \text{kdf}(K_0, 0, 1) \quad (8)$$

Among them, the first parameter in $\text{kdf}(\cdot)$ is the initial key K_0 obtained earlier; the second parameter is the encryption salt, which is a random number and is set to 0 in this protocol; the third parameter is the iteration the number of times, which is set to 1 in this formula.

Next, Bob uses digital compression technology, such as Zigzag, to compress the set $s[\cdot]$ to obtain Cpr_s , and then uses the K_D to generate the corresponding message authentication code, which is calculated as shown in formula (9):

$$\text{HMAC}_{rsp} = \text{hmac}_{md5}(K_D, \text{Cpr}_s) \quad (9)$$

Finally, Bob fills Cpr_s into the Message field, and fills the message authentication code HMAC_{rsp} into the HMAC field. After building the rsp_pkt , Bob sends it to Alice.

- The sender receives the serial number set and generates the initial key. Alice starts a $2T_2$ timer after the T_1 timer ends. Within $2T_2$, if Alice does not receive the rsp_pkt sent by Bob, she abandons the negotiation and reports to the upper layer: the key was not successfully established and the negotiation failed. During this period of time, if Alice receives the rsp_pkt from Bob, she will extract the content of each field. Alice extracts and decompresses the $\text{Cpr}_{s,r}$ to get the set $s[\cdot]_r$ received by Bob. Then, she takes the corresponding random message from the memory according to $s[\cdot]_r$, and calculates the initial key K_0 , the derived key K_D , and the message authentication digest HMAC_{rsp}^l according to formulas (6), (7), (8), and (9).

Next, Alice compares whether the calculated $\text{HMA } C_{rsp}^l$ is the same as the HMAC_{r-rsp} extracted from the rsp_pkt . If not the same, it means that the rsp_pkt was damaged or attacked during transmission, Alice will report to the upper layer: Negotiation failed; If same, it means that the calculated initial key K_0 is also correct.

After the timer of $2T_2$ is over, Alice starts a timer with the time of T_3 again. During this time period, Alice constructs an acknowledgement packet ack_pkt indicating that the initial key K_0 has been generated, and then uses the derived key K_D to calculate the HMAC_{ack} according to the formula (10) and form the ack_pkt and sends it to Bob.

$$\text{HMAC}_{ack} = f_{HMAC}(K_D, \text{RandData}) \quad (10)$$

Then Alice needs to wait for T_3 time again. After the T_3 time ends, Alice enters the waiting phase. At this stage, Alice cannot leave the effective communication range to avoid missing the data packet sent by Bob indicating that the negotiation is over. If during the waiting phase, Alice does not receive the end-of-negotiation packet, even if both parties have established the initial password, the negotiation still fails. The length of the waiting phase is the maximum time that Alice promises Bob, and it can also be set according to the channel conditions, but the premise is that the length of the waiting phase needs to be guaranteed to the maximum.

- Receiver sends negotiation end packet after the $2T_2$ timer expires, Bob starts another timer with a time of $2T_3$. During this period, if Bob receives the `ack_pkt`, he will extract the $HMAC_{r-ack}$, $RandData_r$, and the values of other fields. Then, Bob calculates the $HMAC_{ack}^l$ according to formula (10) based on the K_D and the $RandData_r$. Determine whether $HMAC_{ack}^l = HMAC_{r-ack}$ is true. If it is true, it indicates that Alice has successfully generated the initial key K_0 . After the $2T_3$ timer expires, Bob constructs the end packet and sends it to Alice, indicating that the initial key K_0 was successfully generated. If the HMAC verification equation does not hold or Bob does not receive the confirmation message packet within T_3 , the key negotiation fails.
- Negotiation successful
In the waiting phase, if Alice receives the `end_pkt` and passes the integrity authentication, both communicating parties know that the other party has successfully generated the initial key and the negotiation is successful. Next, both parties can enter the secure communication phase.

3.3 Secure Communication Stage

After obtaining the initial key, Alice and Bob could begin their secure communication. Without loss of generality, we assume that Alice sends messages to Bob. A fully duplex secure communication scheme could be naturally extended from the unidirectional protocol.

This section will introduce the stop and wait mode for self-secure communication. In this mode, each time Alice sends a data packet, she will not send the next data packet until she have received the response packet from Bob. The receiver only needs to process and respond to the received data packets. In order to ensure communication efficiency, the protocol stipulates that Alice and Bob store some information needed for communication in the hard disk and memory, respectively. Meanwhile, during the communication process, Alice will retransmit the data packets which Bob did not receive completely.

For the sender Alice, the content stored in memory contains the following information:

- Sequence number of the packet to be sent Seq_s ;
- The latest communication key K_i ;
- the latest data Msg_i that have been sent and the corresponding ACK_{s-i} ;
- Communication times α_s ;
- The value of the Retran field of the current latest packet; The contents stored on the hard disk include the following:

- The latest key obtained after each W round of communication ends normally is called the W -round key. Store the last two W -round keys $K_{(j-1)*W}$, K_{j*W} ;
- All data and corresponding serial numbers have been sent.

For receiver Bob, the content stored in memory includes the following information:

- The latest communication key K_i ;
- The previous round communication key K_{i-1} ;
- The sequence number of the next expected received packet Seq_r ;
- Communication times α_r ;
- The latest data Msg_{i-1} currently received and the corresponding $ACK_{r.(i-1)}$

The contents stored on the hard disk include the following:

- The latest communication key K_i ;
- The latest key obtained after each W round of communication ends normally is called the W -round key. Store the last two W -round keys $K_{(j-1)*W}$, K_{j*W} ;
- All message and serial numbers that have been completely received;

The communication flow between the two communicating parties is shown in Fig. 4:

First, the communication parties Alice and Bob need to initialize the parameters required for communication. If they are communicating for the first time after negotiating the key, the Seq_s of packets that Alice wants to send and the Seq_r of packets that Bob wants to receive will be initialized to 0; both parties use the key K_0 established during the negotiation phase as the key K_i for this communication.

$$\begin{aligned} Seq_s &= Seq_r = 0 \\ K_i &= K_0 \end{aligned}$$

If this is not the first communication, they will take out the latest packet sequence numbers Seq_s and Seq_r and the communication key K_i stored in memory to initialize the information parameters of this communication. At the beginning of each communication, Alice and Bob set the parameters for recording the number of communication times to 0, that is, $\alpha_s = \alpha_r = 0$.

Next, the two parties began formal communication. First, Alice needs to create a communication packet, The packet construction process is shown in Fig. 5.

She fills the contents of each field in the packet to get the packet $P_{p.s.i}$. According to formula (2), the $P_{p.s.i}$ is filled by the OAEP function to obtain the mediantext packet $P_{m.s.i}$. Then, the key K_i is divided into three parts K_H , K_E , K_A using a key splitting function, to be used to calculate $HMAC_{s.i}$, $P_{c.s.i}$ and $ACK_{s.i}$ respectively.

$$P_{c.s.i} = encrypted_{AES}(K_E, P_{m.s.i}) \quad (11)$$

$$HMAC_{s.i} = f_{HMAC}(K_H, P_{m.s.i}) \quad (12)$$

$$ACK_{s.i} = f_{HMAC}(K_A, exchange(P_{m.s.i})) \quad (13)$$

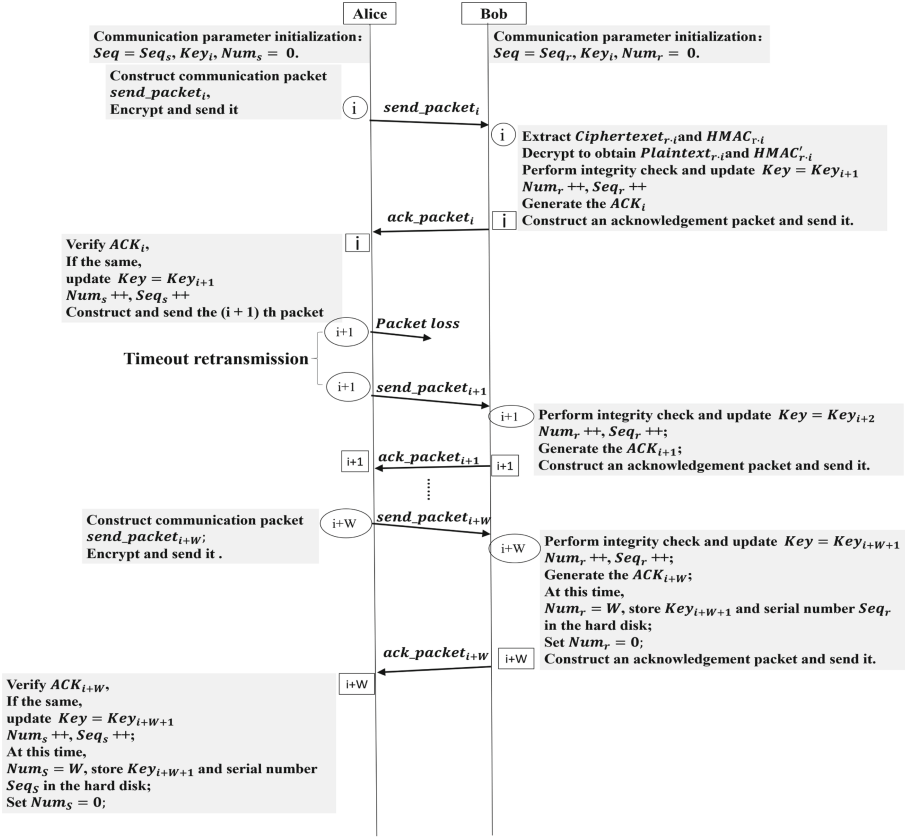


Fig. 4. Stop and wait protocol communication process

Then, the $P_{c,s,i}$ and $HMAC_{s,i}$ are connected to get the send packet $send_pkt$, and the $ACK_{s,i}$ is stored to the specified location in memory. Finally, Alice sends $send_pkt_i$ over the wireless channel and starts the timer at the same time.

For the receiver Bob, he extracts communication messages according to the process shown in Fig. 6.

He extracts $P_{c,r,i}$ and $HMAC_{r,i}$ from the data packet according to the length specified by the protocol. Samely, K_i is divided into three parts K_H, K_E, K_A . Next, according to formulas (6) and (7), Bob uses K_E to decrypt the $P_{c,r,i}$ to obtain the $P_{p,r,i}$, and then further extracts the content of each field according to the length of each field. According to formula (12), the message authentication code $HMAC^l_{r,i}$ is obtained.

Next, Bob performs error checking. If $HMAC^l_{r,i} = HMAC_{r,i}$ Bob can consider that the received data packet is complete. Then, Bob checks whether the Seq_{temp} in the Seq field of the received data packet is not greater than the sequence number Seq_r stored in memory. Bob will perform a parameter update operation. If the equation $Seq_{temp} = Seq_r$ holds, Bob will perform the parameter update operation as follow:

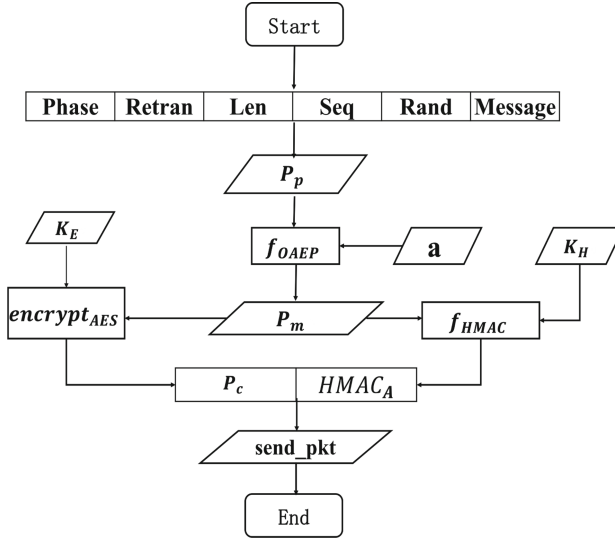


Fig. 5. Packet construction process

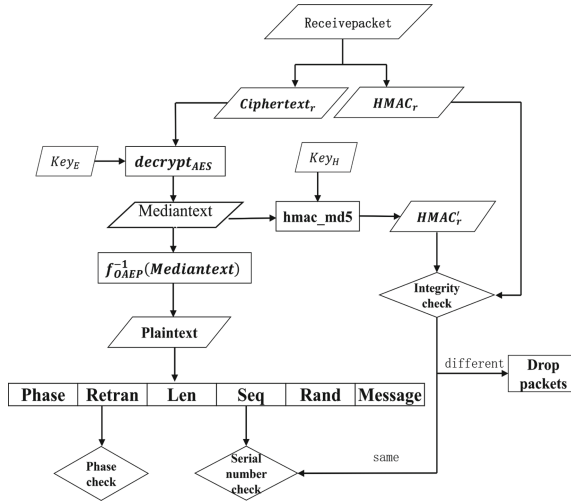


Fig. 6. Packet extraction process

- $Seq_r = Seq_r + 1$;
- According to the formula (13), calculate the acknowledgement message ACK_r based on P_{m-r-i} ;
- According to the different values of the Retran field, Bob updates the key differently. When $Retran = 0$, update the key of this communication according to the following formula (14):

$$K_{i+1} = sha256(K_i \oplus (Msg_i \oplus Rand_i)) \quad (14)$$

When $\text{Retran} = 1$, update the key of this communication according to the following formula (15):

$$K_{i+1} = \text{sha256}(K_i \oplus (\text{Msg}_i \oplus \text{Rand}_i) \oplus (\text{seq}_r - 1)) \quad (15)$$

After the parameter update is completed, Bob will store the K_{i+1} , the communication data, the Seq_{r-1} , and Seq_r to the corresponding location in the memory.

- Determine whether the equation $\alpha_r = W$ is established, that is, whether the current communication is the W th communication. If it is true, store the K_i to the corresponding location on the hard disk, and update the other information stored in the hard disk to the current information, meanwhile set α_r to 0. If it is not true, increase α_r by 1.
- Bob sends a acknowledgment packet with the Seq field set to the updated Seq_r , and the value of the Message field set to $\text{ACK}_{r,i}$.

If $\text{Seq}_r > \text{Seq}_{temp}$, Bob will not perform the update operation, leaving the parameter values in memory unchanged. But Bob still need to calculate the value of $\text{ACK}_{r,i}$ meanwhile sends an acknowledgment packet, setting the Seq field to Seq_r , the value of the Message field to $\text{ACK}_{r,i}$ and the number of communications α_r .

For the sender Alice, if Alice receives the acknowledgment packet from Bob before the timer expires, she will extract the Seq_{ACK} and $\text{ACK}_{r,i}$ in the acknowledgment packet, and check whether the received $\text{ACK}_{r,i}$ is the same as the $\text{ACK}_{s,i}$ stored in the memory. If the same, Alice can determine that the acknowledgment packet was sent by Bob. Then perform sequence number verification.

If $\text{Seq}_{ACK} = \text{Seq}_s + 1$, it means that this is the receiver's confirmation of the currently sent data packet, then Alice will perform a parameter update operation:

- $\text{Seq}_s = \text{Seq}_s + 1$;
- $\alpha_s = \alpha_s + 1$; if $\alpha_s = W$, save the K_i in the hard disk, and set α_s to 0.
- Calculate a new round of communication key K_{i+1} according to formulas (14) or (15), and update the corresponding value in memory;

If $\text{Seq}_{ACK} > \text{Seq}_s + 1$, it means that Bob went to receive the packet whose sequence number is Seq_{ACK} . Alice sets $\alpha_s = \alpha_r$, $\text{Seq}_s = \text{Seq}_{ACK}$, and store Seq_s in memory and Seq_s-1 is stored in the hard disk.

If after the timer expires, Alice has not received Bob's acknowledgment packet, the send packet will be retransmitted and retimed. During the retransmission, the value of the Retran field needs to be set to 1, and the random data is regenerated and filled in the Rand field. Connect α_s to the communication data, and recalculate HMAC and ACK . The values of other fields remain unchanged. If the number of data packet retransmissions exceeds Ω , this communication will end, and Alice reports a communication failure to the upper layer.

4 Experiment

To verify that the proposed protocol is feasible and stable in practical application environments, we use Raspberry 3 to design experiments based on Bluetooth channels, and

use wireshark software installed in a laptop to simulate an attacker to capture packets. The experiment is divided into two parts: verifying the feasibility and the robustness of the protocol. Next, the experimental content and results will be introduced.

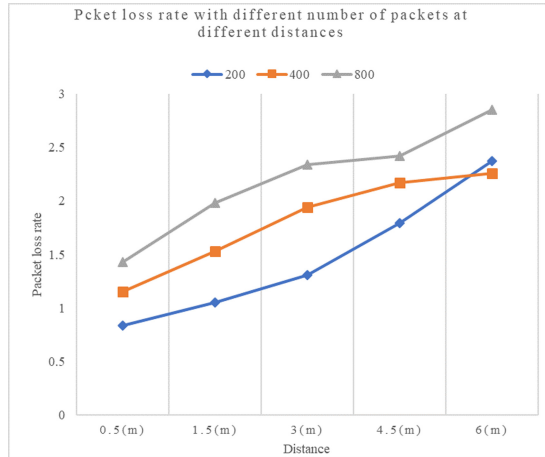


Fig. 7. Pcket loss rate with different number of packets at different distances

– Feasibility experiment

The key agreement algorithm of this protocol uses the characteristics of unavoidable packet loss and error of the wireless channel. Therefore, to verify the feasibility of the protocol, it is necessary to prove that random events such as packet loss and error will occur in the wireless channel. First, we set up a wireless Bluetooth channel between two raspberry 3 s and transfer packets between them. We set the amount of data packets sent by the sender to 200 each time, and the length of the data packet is 64 bytes. The distance between the two parties of the communication is set to 0.5 m, 1.5 m, 3 m, 4.5 m, and 6 m, respectively. Observe the number of correctly and completely received packets by the receiver to calculate the corresponding packet loss rate. Each experiment was repeated 10 times, and the average value were calculated. The sender sends a packet every 10 ms and sends it continuously. Then we changed the number of packets sent again: 400 and 800. The Fig. 7 shows the relationship between the packet loss rate of the wireless Bluetooth channel, the number of packets sent, and the distance. From the Fig. 7 we can observe that as the distance increases, the packet loss rate increases; meanwhile, as the number of sent data packets increases, the packet loss rate also increases to a certain extent. This proves that packet loss events exist in each case. Thus we proved that our protocol is feasible in practical application scenarios.

– Overhead compare

This article also compares the overhead with three key update protocols that have been proposed in the reference. The experimental scenario is that one node in the network is compromised and the key needs to be updated to ensure communication security. We compared the transmission overhead, calculation overhead, and space overhead,

Table 2. Overhead comparison

Update algorithm	Transmission overhead	Calculation overhead	Space overhead
[2]	$O(n)$	$O(n)$	$O(n)$
[4]	$O(n)$	$O(1)$	$O(\log n)$
[19]	$O(\log n)$	$O(n)$	$O(\log n)$
Propose	0	$O(n)$	$O(n)$

as shown in the Table 2. We have found that, compared with other protocols, the computational and space overheads of this protocol are relatively large, but because no key or key material needs to be transmitted, the transmission overhead is 0. Compared with other protocols, it has a great advantage in the limited bandwidth and highly dynamic IoT scenarios.

5 Conclusion

This paper proposed a self-secure communication scheme to address the challenges in IoT secure communications. The scheme contributes new ideas in the key distribution and the key update processes, so that IoT device nodes could establish secure communication without relying on the pre-shared key materials or the inter-operable public key infrastructure. More importantly, the proposed scheme is resilient to the key theft attacks. The scheme would force the stolen key to become invalid as the communication goes on. The IoT device nodes do not need to intervene and the communication security is recovered. This paper further illustrates a fail-safe implementation strategy for the proposed scheme that allows the scheme to be readily converted into an engineering solution.

References

1. Ashton, K.: That ‘Internet of Things’ thing. *RFID J.* **101-1** (2009)
2. Xia, F., Yang, L.T., et al.: Internet of Things. *Int. J. Commun. Syst.* **25**, 1101–1102 (2012)
3. Alabaa, F.A., Othmana, M., et al.: Internet of Things security: a survey. *J. Netw. Comput. Appl.* **88**, 10–28 (2017)
4. Romana, R., Alcaraza, C., et al.: Key management systems for sensor networks in the context of the Internet of Things. *Comput. Electr. Eng.* **37**, 147–159 (2011)
5. Jie, C., Liang, Y.-C., et al.: Intelligent reflecting surface: a programmable wireless environment for physical layer security. *IEEE Access* **7**, 82599–82612 (2019)
6. Pinto, T., Gomes, M., et al.: Polar coding for physical-layer security without knowledge of the eavesdropper’s channel. In: 2019 IEEE 89th Vehicular Technology Conference (VTC 2019-Spring), pp. 1–5, IEEE, Kuala Lumpur (2019)
7. Xiang, Z., Yang, W., et al.: Physical layer security in cognitive radio inspired NOMA network. *IEEE J. Sel. Top. Sig. Process.* **13**(3), 700–714 (2019)
8. Melki, R., et al.: A survey on OFDM physical layer security. *Phys. Commun.* **32**, 1–30 (2019)
9. Shen, J., Moh, S., et al.: A novel key management protocol in body area networks. In: ICNS 2011: The Seventh International Conference on Networking and Services, pp. 246–251 (2011)

10. Li, Y.: Design of a key establishment protocol for smart home energy management system. In: 2013 Fifth International Conference on Computational Intelligence, Communication Systems and Networks, pp. 88–93. IEEE, Madrid (2013)
11. Sciancalepore, S., Capossole, A., et al.: Key management protocol with implicit certificates for IoT systems. In: IoT-Sys 2015, Proceedings of the 2015 Workshop on IoT Challenges in Mobile and Industrial Systems, pp. 37–42. Association for Computing Machinery, New York (2015)
12. Saied, Y.B., Olivereau, A.,: D-HIP: a distributed key exchange scheme for HIP-based Internet of Things. In: 2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), pp. 1–7. IEEE, San Francisco (2012)
13. Riyadh, M., Affiliated, A., Djamel, T.: A cooperative end to end key management scheme for e-health applications in the context of Internet of Things. *Ad-hoc Netw. Wirel.* **8629**, 35–46 (2015)
14. Porambage, P., Braeken, A., et al.: Proxy-based end-to-end key establishment protocol for the Internet of Things. In: 2015 IEEE International Conference on Communication Workshop (ICCW), pp. 2677–2682. IEEE, London (2015)
15. Veltri, L., Cirani, S., et al.: A novel batch-based group key management protocol applied to the Internet of Things. *Ad Hoc Netw.* **11**(8), 2724–2737 (2013)
16. Abdmeziem, M.R., Tandjaoui, D., et al.: A decentralized batch-based group key management protocol for mobile Internet of Things (DBGK). In: 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, pp. 1109–1117. IEEE, Liverpool (2015)
17. Jing, Q., Vasilakos, A.V., Wan, J., Lu, J., Qiu, D.: Security of the Internet of Things: perspectives and challenges. *Wirel. Netw.* **20**(8), 2481–2501 (2014). <https://doi.org/10.1007/s11276-014-0761-7>
18. He, X., Niedermeie, M., et al.: Dynamic key management in wireless sensor networks: a survey. *J. Netw. Comput. Appl.* **36**(2), 611–622 (2013)
19. Varalakshmi, R., Uthariaraj, V.R.: Huddle hierarchy based group key management protocol using gray code. *Wirel. Netw.* **20**(4), 695–704 (2013). <https://doi.org/10.1007/s11276-013-0631-8>
20. Conti, M., Dehghantanha, A., et al.: Internet of Things security and forensics: challenges and opportunities. *Future Gener. Comput. Syst.* **78**, 544–546 (2018)
21. Al-Sarawi, S., Anbar, M., et al.: Internet of Things (IoT) communication protocols: review. In: 2017 8th International Conference on Information Technology (ICIT), pp. 685–690. IEEE, Amman (2017)
22. Li, Y.: Design of a key establishment protocol for smart home energy management system. In: 2013 Fifth International Conference on Computational Intelligence, Communication Systems and Networks, Madrid, , pp. 88–93 (2013)
23. Nguyen, K.T., Laurent, M., et al.: Survey on secure communication protocols for the Internet of Things. *Ad Hoc Netw.* **32**, 17–31 (2015)
24. Abdmeziem, M.R., Tandjaoui, D.: An end-to-end secure key management protocol for e-health applications. *Comput. Electr. Eng.* **44**, 184–197 (2015)