RSA-Based Handshake protocol in Internet of Things

Junye Mao , Huiying Zhu , Yulan Liu , Yuanjing Liu , Weihao Qian , Jie Zhang , Xin Huang
Department of Computer Science and Software Engineering
Xi'an Jiaotong-Liverpool University
Suzhou , China
Junye.Mao16@student.xjtlu.edu.cn , Huiying.Zhu16@student.xjtlu.edu.cn,
Yulan.Liu16@student.xjtlu.edu.cn , Yuanjing.Liu16@student.xjtlu.edu.cn
Weihao.Qian16@student.xjtlu.edu.cn , Xin.Huang@xjtlu.edu.cn

***Abstract* — Potential security vulnerabilities exist in network layer under the environment of Internet of Things(IoT). Attacks could jeopardize the commutation between devices if an encryption algorithm is not applied. In order to address this issue, we design a RSA-based handshake protocol which implements a secure communication interface in the network layer. In such case, both the controller and device can verify the other's identity and a new session key is generated for subsequent communication. However, several experiments indicate that this protocol costs substantial time to process with RSA. The evaluation results show that several improvements should be completed in the algorithm.**

***Index Terms — RSA, Wireless reprogramming, Internet of Things, Handshake protocol, Industrial 4.0*.**

I. INTRODUCTION

The Internet of Things(IoT) is considered to have a revolutionary impact on modern industry fueled by emerging technologies, such as wireless sensing and mobile communication. In addition to its tremendous contribution in industrial sectors, IoT is expanding the scope of application in smart transportation, healthcare, environmental monitoring, etc [1]. According to the three-layer architecture, IoT is divided into perception layer, network layer and application layer [2].

A notable challenge appears to lie in the network layer in IoT in terms of operating and maintaining numerous devices. As a matter of fact, the devices are constantly required to be reprogrammed through wireless connection in consideration of improving security and functionalities. Thus, security vulnerabilities exploited by attackers in the wireless reprogramming process could jeopardize the communication between the controller and the device if an encryption algorithm is not applied. In such case, an unauthorized eavesdropper may participate in the communication and cause tampering or leakage of information. Therefore, we have to make sure that the recipient receives legal and correct messages. The report proposes a RSA-based handshake protocol based on Software Definition Function (SDF) in order to guarantee a secure wireless communication between devices.

RSA encryption algorithm is a public-key cryptography that can be used for both encryption and digital signature [3]. It generates a pair of keys known as the public key and the private key for encryption and decryption purposes respectively to ensure confidentiality in communication and guarantees authentication and integrity through digital signature. Therefore, the communication will not be interfered by unauthorized intruders [3].

The rest of the paper is organized as follows. Section 2 provides detailed information of IoT together with theoretical basis of RSA encryption algorithm. Section 3 focuses on the analysis of handshake protocols. In Section 4, a number of experiments are conducted and a critical evaluation is presented based on the testing results. In section 5, the proposed protocol is compared with other Handshake protocols based on SDF. Finally, the conclusion is drawn in section 6.

II. BACKGROUND

In order to thoroughly understand the protocol, background information about IoT and RSA encryption algorithm is presented.

A. IoT Overview

Typically, IoT can be divided into two parts. One contains many entities, including physical devices and software [1]. The other is the data, which means entities can exchange data with each other. Based on the three-layer architecture, IoT includes three layers: 1) the perception layer which collects information via wireless sensor networks; 2) the network layer consisting of access network and transmission network which safely transmits the gathered information to the application layer; and 3) the application layer which processes data to realize control and management of entities in the real world [1].

B. RSA Foundation

RSA is a popular public key encryption algorithm which can be used for both encryption and digital signature invented by Ron Rivest, Adi Shamirh and LenAdleman in 1977. It generates a freely distributed public key and a secret private key for encryption and decryption purpose respectively. Its security rests on the presumed difficulty in terms of large number factorization and proves to be relatively secure for it resists all known password attack to date. Since 1992, it has been the public key data encryption standard recommended by ISO.

1) Generation of Key Pair

The public key can be represented as (E, N) and the private key as (D, N) [4]. The table below illustrates the computation of E, N and D [4].

TABLE 1. PRINCIPLE OF KEY PAIR

N	N=p * q; p and q are prime numbers.
L	L = lcm (p-1, q-1); L is the least common multiple of p-1 and q-1.
E	1<E<L, gcd (E, L) = 1; E is a random integer between 1 and L, the greatest common divisor of E and L equals 1.
D	1 <D <L; E * D mod L=1. D is a random integer between 1 and L, the remainder of E*D equals 1.

2) Encryption and Decryption

The encryption and decryption processes are presented in the figure 1. (m=plaintext, c=cypher text)

TABLE 2. PROCESS OF ENCRYPTION AND DECRYPTION

Encryption	$c = m^E \bmod N$
Decryption	$m = c^D \bmod N$

Fig. 1. Encryption and Decryption process [5]

3) Digital Signature

A digital signature ensures the authentication, non-repudiation and integrity of the message sent to the recipient. The signing process can be concluded as: 1) The sender *A* extracts the digest *h(m)* of message *m* by using a hash function, and use its private key (D, N) to encrypt *h(m)* to generate the signature *s*(s=h(m)*D mod N); 2) *A* uses the public key of the recipient *B* to encrypt *s* and *m* and send the cypher text *c* to *B*. Once B receives *c*, it carries out the following steps to verify the signature: 1) *B* uses its private key to decrypt *c* and acquires message *m* and signature *s*; 2) *B* uses A's public key to decrypt *s* and acquires *H(m)*; 3) *B* uses the same hash function to extract the digest *h(m)*; 4) *B* compares *h(m)* with *H(m)*. The verification only succeeds when *h(m)* equals *H(m)*.

III. PROTOCOL

A RSA-based handshake protocol is designed to implement the secure communication in network layer which is illustrated in figure 2. After a successful run of this handshake protocol, the controller can guarantee that the received messages are sent by the verified device and they have not been leaked or tampered. Meanwhile, the device receives messages sent by the verified controller and they have not been leaked or tampered [6]. Furthermore, in order to achieve security goals, a new session key *K* is generated for encryption in the subsequent communication. The experiment and test of the protocol are presented in Section 4.

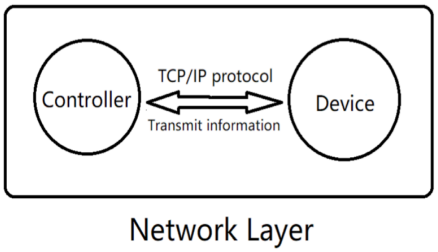

Fig. 2. Protocol in the network layer

A. Protocol Review

As is discussed above, the controller and device shall share their public keys with each other and the private keys are held by themselves to decrypt encrypted messages in RSA. Accordingly, *PK_C* and *SK_C* represent the public key and private keys of the controller, and *PK_D* and *SK_D* denote those of the device. *FEATURE_REQUEST* and *FEATURE_REPLY* denote the feature of the controller and the device respectively for purpose of identity verification under special circumstance.

1) Establishment of TCP Connection

Before the TCP connection is established, the controller first passively opens its familiar port and keeps monitoring this port. When the device tries to connect with the controller, it initiates a request to open the temporary port. After that, the controller and device perform three handshakes with TCP/IP to establish a secure network connection and then transmit information. If the connection is successfully built, the controller will print out a prompt hint message.

2) Verification Process and Key Generation

The flow of messages is illustrated in figure 3 and the protocol is presented below.

First, the device sends a *msg1* to socket which consists of a hello message, a random large number *nonceND* ranging from 10000000 to 999999999 and its numerical identity *D_ID*. Meanwhile, the device sends *msg2* to the port which consists of text "hello_c", a random large number *nonceND* and its numerical identity number *C_ID*.

$$msg1 = (hello\_msg||nonceND||D\_ID)$$

$$msg2 = (hello\_msg||nonceNC||C\_ID)$$

Then the controller and device receive *msg1* and *msg2* respectively from the established port. The controller generates a random temporary master key *TMK* ranging from 10000000 to 999999999 and encrypts it along with *FEATURE_REQUEST* using *PK_D*. After that, the controller generates the signature *SIG_C* by computing the digest of string *m* which consists of *msg1, msg2* and *FEATURE_REQUEST* using SHA-256 algorithm, and encrypts the digest *m* with *SK_C*. The controller sends *msg3* consisting of encrypted *FEATURE_REQUEST*, encrypted *TMK* and encrypted *SIG_C* to the port.

$$SIG\_C = (msg1||msg2||FEATURE\_REQUEST)_{SK\_C}$$

$$msg3 = SIG(FEATURE\_REQUEST_{PK\_D}||\ TMK_{PK\_D}||SIG\_C)$$

Upon receiving *msg3,* the device verifies *SIG_C*. It first initializes a verification string which equals to *m* and uses SHA-256 algorithm as well to compute its digest n. Then it decrypts *msg3* using *SK_D* to acquire *SIG_C* and uses *PK_C* to decrypts *SIG_C* to acquire *m*. Verification succeeds only when m=n. If so, the device then computes its signature denoted as *SIG_D* with the same method introduced above. Then it uses *SK_D* to decrypt *msg3* to acquire *FEATURE_REQUEST* along with *TMK*, and uses *TMK* to compute the shared session key *K* through SHA-256 algorithm and prints out *K*. After that, it encrypts *FEATURE_REPLY* using *PK_C* and sends *msg4* consisting of encrypted *FEATURE_REPLY* and *SIG_D* to the controller.

$$SIG\_D = SIG(msg1||msg2||FEATURE\_REQUEST||FEATURE\_REPLY)_{SK\_D}$$

$$msg4 = (FEATURE\_REPLY_{PK\_C}||SIG\_D)$$

After receiving *msg4*, the controller carries out the same verification process of *SIG_D* as *SIG_C*. If it succeeds, the device then decrypts msg4 with SK_C to acquire *FEATURE_REPLY*. The session key *K* is computed with the same method introduced before.

Above all, *FEATURE_REPLY* and *FEATURE_REQUEST* is transmitted after being decrypted with RSA algorithm. It will strengthen the security level of feature information to prevent it from being spied by eavesdropper. The *SIG_D* and *SIG_C* are signatures signed with RSA which can verify other devices and protect the key information from falsification. Another fresh key k is generated for the subsequent communication.

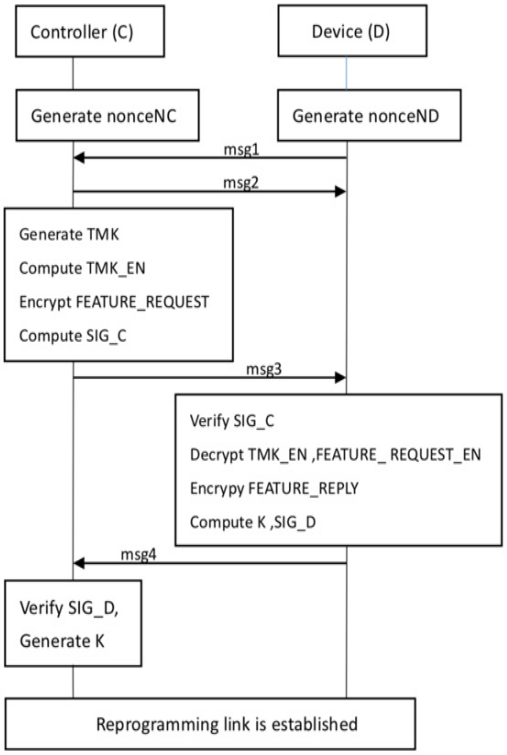

Fig. 3. Handshake protocol based on RSA

IV. Experiment and Evaluation

A. Theoretical evaluation

The security of RSA depends on the difficulty of large number factorization. The key point of attacking RSA system is factorizing n. If the factorization is successful, n=p*q can be calculated. Thus, in order to keep RSA secure, p and q must be sufficiently large primes so that analysts are unable to factorize n in polynomial time. To resist the existing integer factorization algorithm, the prime factors p and q of the RSA modulus n require that p-1 and q-1 respectively contain large prime factors p_1 and q_1, p_1-1 and q_1-1 respectively contain large prime factors p_2 and q_2, and p+1 and q+1 respectively contain large prime factors p_3 and q_3 [7]. Figure 4 indicates the time required for factorization of an integer with different bit length.

TABLE 3. Time Of Factorization

Key size/bit	MIPS/year
512	3×10^4
768	2×10^8
1024	3×10^{11}
2048	3×10^{20}

Fig. 4. Time required for factorization of integer [8].

As long as the length of n meets certain requirements, the parameters p, q and e are chosen properly, the RSA system is quite secure. However, the time required for factorization increases exponentially as the key length increases [8]. The process of encryption will also take substantial amount of time for a long text.

B. Experiment Result

As is shown in figure 5 and 6, the controller and the device could connect successfully. The corresponding messages and features are printed out in the interface.

Fig. 5. Interface in controller

Fig. 6. Interface in device

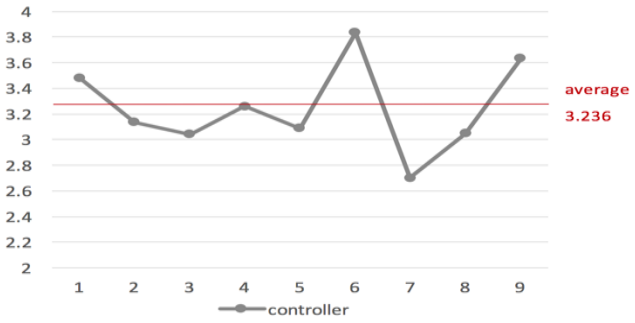

Fig. 7. Average run time of controller

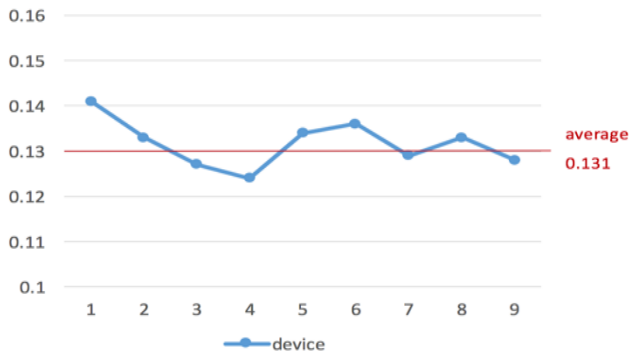

Fig. 8. Average run time of device

As is illustrated in figure 7 and 8, the run time is prolonged and the average run time is approximately about 0.131 seconds and 3.236 seconds in devices and controllers respectively. That means it will cost long time for controller to verify and establish connection with device. The standard deviations of run time are 0.324 seconds and 0.005 seconds respectively, which shows stabilization. However, in actual environment, the time of each communication step of two parties will become longer due to distant distance between controllers and devices. In IoT environment, it is not user-friendly for client to spend substantial time on waiting for controller's response.

Feature request and feature reply are encrypted in the protocol. The data would not be obtained by the eavesdropper during the transmission, which ensures the security. It realizes confidentiality of the identity features through communication in IoT, which excludes the possibility of third party interference and theoretically improves security.

However, when it comes to time complexity, it takes a long time to establish secure communication between devices, which degrades the performance and affects user experience. The RSA public key cryptosystem involves a large number of numerical computation in the encryption or decryption processes. It is noticeable that efficiency of decryption is lower than that of encryption. The decryption time and the size of the decrypted file show a linear growth trend.

V. Related Work

In this section, several related works will be presented briefly to compare with the RSA-Based handshake protocol in theory.

A. Overview of related work

A ECDH-based handshake protocol for the SDF wireless updating introduced by *J.Cai et al.* is shown in the following [9].

The protocol named unbalanced Open-Function handshake protocol is based on the Open-Function handshake protocol and utilizes Elliptic Curve Cryptography (ECC) algorithm in SDF framework to guarantee the security of authentication. This protocol is to establish communication, transmit update files and reprogram end devices. It intends to increase efficiency by reducing the computational load on function station and provides security of communication.

B. Comparison

The corresponding features between the proposed protocol and the related protocol is compared in Table 4. The proposed protocol utilizes RSA algorithm to increase security and the unbalanced Open-Function uses ECC algorithm to increase efficiency.

In the process of handshake, the proposed protocol encrypts the feature request and feature reply with RSA algorithm to avoid leakage of key information. Two participants could verify the identification both in the sever and device because the feature data will be presented in user interface and participants could affirm this information.

In addition, in terms of authentication, it uses SHA-256 as the hash algorithm because the SHA-1 algorithm family was officially deprecated by NIST in 2011 [10]. Moreover, SSL labs suggested that the SHA-1 algorithms should be replaced with SHA-2 algorithm [11]. Therefore, the security of the protocol is higher than protocols applied with SHA-1 algorithm.

TABLE 4. Feature Comparison of Proposed Protocol and Its Predecessors ("√" denotes the protocol possesses the feature, and "X" denotes the protocol does not own this feature)

Feature	**Unbalanced Open-Function protocol**	**RSA-Based handshake protocol**
Employing ECC algorithm in authentication	√	X
Employing RSA algorithm in authentication	X	√
Encrypted feature data	X	√
Number of handshake steps	4	4
Protocol description and detail	√	√

VI. Conclusion

This report has critically come up with a handshake protocol which can establish reprogramming link and generate fresh communication key between two networking devices. This protocol can prevent devices' messages and features data from being falsified and tampered. According to the experiment, time consumption of the protocol is substantial. Future study could improve the efficiency by searching in the mixed algorithms.

Acknowledgment

This work was supported by the XJTLU research development fund projects under Grant RDF140243 and Grant RDF150246, in part by the National Natural Science Foundation of China under Grant No. 61701418, in part by Innovation Projects of The Next Generation Internet Technology under Grant NGII20170301, in part by the Suzhou Science and Technology Development Plan under Grant SYG201516, and in part by the Jiangsu Province National Science Foundation under Grant BK20150376.

References

[1] S. Li et al., "The Internet of Things: A Survey," *Inform. Syst. Front.*, vol. 17, no. 2, pp. 243-259, Apr. 2015.

[2] P. Sethi and S. R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications," *J. Electr. Comput. Eng.*, vol. 2017, pp. 1-25, Jan. 2017.

[3] R. Rivest *et al.*, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Commun. ACM.*, vol. 21, no. 2, pp.120-126, Feb. 1978.

[4] S. Mahajan and M. Singh, "Analysis of RSA algorithm using GPU programming," *Int. J. Network. Secur. Appl.*, vol. 6, no. 4, Jul. 2014.

[5] B. Schneier, Applied cryptography, second edition, NY: John Wiley & Sons, Inc., 1996.

[6] K. T. Nguyen et al., "Survey on secure communication protocols for the Internet of Things," *Ad. Hoc. Netw.*, vol. 32, pp. 17-31, Sep. 2015.

[7] M. Bakhtiari and M. A. Maarof, "Serious Security Weakness in RSA Cryptosystem," *Int. J. Comput. Sci. Iss.*, vol. 9, no. 3, pp. 175-178, Jan. 2012.

[8] S. Cavallar *et al*., "Factorization of a 512-bit RSA modulus," Rep. MAS-R0007, 2000.

[9] J. Cai *et al.*, "Handshake Protocol with Unbalanced Cost for Wireless Updating," *IEEE Access*, vol. 6, pp. 18570-18581, Mar. 2018.

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

[11] I. Ristic, "SSL/TLS deployment best practices," vol. 1.3, pp. 4-5, Sep. 2013.