Session Key Agreement for End-to-End Security in Time-Synchronized Networks

Qinghua Wang[1], Xin Huang[2], Dawit Mengistu[1]
[1]Department of Computer Science, Faculty of Natural Science, Kristianstad University, SE-29188 Kristianstad, Sweden
[2]Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China
qinghua.wang@hkr.se; xin.huang@xjtlu.edu.cn; dawit.mengistu@hkr.se

Abstract— **Time synchronization is important for many network applications. This paper utilizes the fact that time service is a standard service in almost all kinds of computer networks, and proposes a new session key agreement protocol by building sessions keys based on the locally available time information. A prototype system has been tested in the simulation environment and the results are promising.**

Keywords— Session key; time synchronization; security

I. Introduction

In the past decade, many new types of network technologies have emerged, such as wireless sensor networks [1], Internet of Things [2], cloud computing, fog computing, etc. The Internet is expanding with more and more network devices and there are also a tremendous number of new network applications. Many of these network applications are security sensitive but the current level of protection is not adequate. Reasons are many but mainly due to the using of low cost network devices (e.g. in personal area networks), the lack of standardization for new network types, and the versatility of network applications.

Low cost means a limited computing, memory and communication resources, which also prohibit the implementation of complex security functionalities, such as the RSA algorithm [11] which performs exponentiation operations on large numbers and requires lots of resources. Low cost also often means the lack of security consideration. Actually, many cheap personal devices have become one of the largest security risk for home networks as they allow attacks to be launched from internal networks once these devices are compromised.

Standardization is lagging behind for new types of networks. There are many proposals for wireless sensor networks and Internet of Things. The Zigbee standard [3], the Bluetooth LE [4], and the 802.11ah [13] are a few examples. But none of them is dominant. There are also many proprietary network protocols. This not only creates problems for interconnection, but also raises security concerns as the resources required to defend many different types of devices are much more than that required to defend a single or few types of devices.

There are a variety of new network applications. Some of these applications are security sensitive, such as the healthcare system and the traffic system. Some of these might not sound very sensitive, such as a smart home application, where room temperature is constantly measured. Developers tend to ignore the security aspects if the applications are considered to be non-sensitive. The problem is that once a non-protected network device is compromised, it could be used as an attack vector to attack other sensitive parts of the network. One example is that most routers allow the administrator login from an internal network. In this case, if a home appliance is compromised, it can be later used to change the settings of home routers and allow the monitoring of home networks.

This paper proposes a new session key agreement protocol using the time information which is easily available in most kinds of network devices. The new session key agreement protocol avoids complex handshake process, and can be used to enhance the security of occasional on-demand network traffic which has been a pattern in many new types of network applications (e.g. the Internet of Things).

II. Time Synchronization

Time synchronization is a fundamental service for many network applications, where it is important for the distributed system components or network nodes to maintain the same view on time. Most network nodes on the Internet are relying on the Network Time Protocol [5] to synchronize their local times with a global reference. Detached networks such as wireless sensor networks have their own ways of time synchronization which help all sensor nodes to synchronize with each other (but not necessarily to a global reference).

Actually, accurate time synchronization is not only a desired feature of many network applications, but is also important for infrastructure operation and system services. For example, the time-division multiple access (TDMA) protocols only work in time-synchronized networks [14], the energy-saving duty cycling (or sleeping) strategy only works when peer nodes agree on when to wake up, and a ranging technique using radio signals needs to measure the time that it takes to send a signal from a sender to a receiver, etc.

In this paper, we are assuming a time-synchronized network where all peer nodes have been synchronized with each other up to a certain precision. A high-precision time synchronization, compared to a low-precision one, will allow more bits in the time value to be utilized in the key agreement process, and thus more resilient to a guessing attack. It should, however, be noticed that the confidentiality of network communication in the proposed protocol is not relying on the predictability of a time value. The role of the time information is to guarantee the freshness of a message, and thus largely replaces the role of a *nonce* in many traditional key agreement protocols [7]. After this paper has already been drafted, it is found that the authors in [15] also briefly mentioned the possibility of using a reliable clock as an alternative way to generate *nonce*s, though no practical solution was presented.

III. A New Session Key Agreement Protocol

In this section, we are proposing a new session key agreement protocol. Before we do that, we need to assume that the participating nodes have already shared a secret, which is called the pairwise master key (PMK) in this paper. The *PMK* can be a preset password or a symmetric key which is agreed online via a key distribution protocol (e.g. the Diffie-Hellman protocol [6, 11]).

In the widely implemented Wi-Fi Protected Access II (i.e. WPA2, or IEEE 802.11i standard) [7], a *nonce* (which is a random number) is exchanged among parties every time when a new session key is established. The purpose of using a *nonce* is to update the old session key, guarantee freshness of the new connection, and thus prevent attacks such as a replay attack. However, the exchange of *nonce*s cause extra delay and overhead, and may suffer from packet losses as well. The Wi-Fi WPA2 security protocol adopts a 4-way handshake method to exchange *nonce*s. A very recent research, however, shows that the 4-way handshake is vulnerable to the Key Reinstallation attack [8], which is a severe replay attack and could eventually lead to the decryption of encrypted messages.

Our protocol uses a network node's pre-synchronized clock value to replace the role of a *nonce* and does not require a handshake protocol to exchange parameters. Therefore, our protocol is immune to the newly discovered Key Reinstallation attack. In comparison to WPA2 or any handshake-based security negotiation, our protocol is more light weight and allows instant secure communication. The details of the new session key agreement protocol is shown below:

***Step* 1**: **Sender** queries its local time and fetches the part (depending on the synchronization precision) that is synchronized with the **Receiver**. The synchronized time is denoted as *t*;

***Step* 2**: **Sender** generates a random number, and denotes it as *salt*;

The using of a *salt* guarantees the uniqueness of every session key even in the occasional situation when some time values are repeated (e.g. during a burst transmission, due to daylight saving time, after computer reboot, when a time synchronization protocol is attacked, or when the clock readings are directly manipulated [15]). The *salt*, however, needs not to be long to guarantee security.

***Step* 3**: **Sender** generates a dynamic session key *PTK* based on the knowledge of the local time *t*, *salt*, the master key *PMK*, and the addresses of the sender and the receiver.

PTK = HMAC (*PMK*, *sender_addr* || *receiver_addr* || *t* || *salt*);

A cryptographic hash algorithm HMAC [10, 11] is used here on generating *PTK*. In our prototype implementation, we have adopted the HMAC-SHA-384 variant as it is deemed secure and its output length fulfills our requirement.

Two keys are extracted from *PTK*, and they are the one-time encryption key *OEK* and the one-time authentication key *OAK*. In our prototype implementation, the first 128 bits of *PTK* are used as *OEK*, and the 128 bits following after *OEK* are used as *OAK*. It is also possible to use longer keys.

***Step* 4**: **Sender** encrypts the message to be sent with *OEK*.

ciphertext = AES (*OEK*, *plaintext, encryption mode*);

On the above, *plaintext* is the unencrypted message to be sent and we have adopted the AES algorithm [11, 12] on encryption.

***Step* 5**: **Sender** calculates the message authentication code using OAK.

mac = HMAC (*OAK*, *ciphertext*);

On the above, *mac* stands for the message authentication code. Again, a cryptographic hash function is used here. In our prototype implementation, we have adopted HMAC-SHA-1 as the HMAC algorithm because its hash output is relatively short and it is secure enough for session authentication purpose. The using of *ciphertext* instead of *plaintext* as input makes it possible to detect a message reception error without decrypting the *ciphertext* first.

***Step* 6**: **Sender** sends the *ciphertext* in together with *mac* and *salt* to **Receiver**. Both *mac* and *salt* are sent as envelope information. This is the only message that requires to be sent in this session key agreement protocol.

***Step* 7**: On receiving the message, **Receiver** queries its local time and fetches the part (depending on the synchronization precision) that is synchronized with the **Sender**. The synchronized local time is denoted as *t'*.

***Step* 8**: **Receiver** generates a dynamic session key *PTK'* based on the knowledge of the local time *t'*, *salt*, the master key *PMK*, and the addresses of the sender and the receiver.

PTK' = HMAC (*PMK*, *sender_addr* || *receiver_addr* || *t'* || *salt*);

***Step* 9**: **Receiver** extracts the one-time decryption key *OEK'* and the one-time authentication key *OAK'* from *PTK'* as in Step 3.

***Step* 10**: Receiver verifies the message authentication code using *OAK'*.

mac' = HMAC(*OAK'*, *ciphertext*);

If *mac'* is the same as the *mac* in the received message, the message is authenticated and **Receiver** continues to Step 11.

If *mac'* is not the same as the received *mac*, it could mean either *OAK'* or the received *ciphertext* (including envelope information) is not correct. In either case, **Receiver** cannot correctly decrypt the message and must abort the session. As a remedy, a negative acknowledgement could be sent back to **Sender** indicating the necessity of message retransmission.

***Step* 11**: Receiver decrypts the received message with *OEK'*.

plaintext = AES (*OEK'*, *ciphertext*, *decryption mode*);

Because *OAK'* has been verified to be correct in Step 10, and *OEK'* is generated in the same way as *OAK'*, we know *OEK'* is also correct and the original *plaintext* will be recovered.

The proposed session key agreement protocol guarantees each time a new message is sent, a different encryption key is used. There is no extra overhead on key agreement by removing the need of handshake. The only prerequisite is that there is time synchronization which is already a default service nowadays in many types of networks. Though not all types of network communications would prefer a new session key for every new message, the proposed scheme should fit well for low-data-rate applications, e.g. wireless sensor networks [1] and the Internet of Things [2].

IV. Prototype Implementation and Results

We have made a prototype implementation in Matlab [9] to verify the functionality of the proposed key distribution scheme. More concretely, we have implemented a Sender module and a Receiver module. Every time a new message is sent, the Sender calculates a temporary key following the Steps 1-3 in the key distribution protocol. Our implementation has used the HMAC-SHA-384 algorithm in Step 3 which results in a temporary key of 384 bits. Part of the temporary key (which is the first 128 bits in our implementation) is used as the AES encryption key to encrypt the message. Another part of the temporary key (which has also 128 bits but ranges from bit position 129 to bit position 256) is extracted as the message authentication key. The message authentication code is then created with the HMAC-SHA-1 algorithm. For the Receiver, source address, destination address, *salt*, and the message authentication code are part of the envelope information, and the master key is a pre-shared secret. The only requirement for the Receiver to be able to calculate a temporary key is its local time. In order to simulate the clock difference between the Sender and the Receiver (due to time synchronization error and transmission delay), we manually introduce a time delay (which is a tunable parameter) in the source code before calling the message reception function in the Receiver module.

Matlab allows us to read the current date and time using the *now()* function which returns a serial date number representing the whole and fractional number of days from a fixed, preset date (January 0, 0000). This allows us to easily integrate the time information into the cryptographic function by rounding the time value to a desired precision (i.e. number of digits after the decimal point). For example, the usable time information for encryption and decryption would include all integer digits and one fractional digit to the right of the decimal point if the synchronization precision is 0.1 day.

TABLE I. Time Synchronization Precision vs. Error Rate

Synchronization Precision	**Error Rate (Clock difference = 10s)**	**Error Rate (Clock difference = 1s)**
1 day	0	0
0.1 day	0	1%
0.01 day	1%	0
0.001 day	14%	3%
0.0001 day	100%	21%
0.00001 day	100%	100%

We have simulated the secure communication with different experimental setups and the results are shown in Table I. We have made 100 tests for each unique experimental setup. It can be seen that the error rate (i.e. when the Receiver uses a wrong key to decrypt a message) is dependent on the assumed time synchronization precision as the assumed time synchronization precision determines how many digits are used to represent the current time in encryption and decryption, and is also dependent on the actual clock difference between the Sender and the Receiver. When the actual clock difference between the Sender and the Receiver is smaller than one unit of the assumed time synchronization precision (i.e. when there is a correct assumption on the synchronization precision), the error rate is generally small or 0. For example, when the synchronization precision is 0.01 day (which is equivalent to 864 seconds), the receiver error rate is 1% for a clock difference of 10 seconds (which is smaller than 864 seconds), and 0 for a clock difference of 1 second (which is also smaller than 864 seconds). But if the clock difference is larger than one unit of the assumed time synchronization precision (i.e. when the assumed synchronization precision cannot be achieved in practice), the system is not working as the error rate goes up to 100%. Therefore, it is important not to assume a higher time synchronization precision than what can actually be achieved in the communication system.

The small error rates which have been experienced even for relative low synchronization-precision are due to round-up errors of the local times. This error can be easily detected via verifying the message authentication code. On verifying the message authentication code at the Receiver, a message authentication key which is also generated (partially) from the time information is used. Therefore, there will be a failure on verifying the message authentication code if the rounded-up time is different from that of the Sender. In order to fix this problem, a message retransmission can be used. Because a retransmitted message is happening at a new time and thus a different rounded-up time, the chance of a failed retransmission (due to the mismatch between the rounded-up times at the Sender and at the Receiver) will be low.

V. Supplementary Results

One more interesting experiment has been done after the draft has been submitted. The results are added here as supplementary results.

In the previous section, we saw the effectiveness of using time information in securing communication sessions, but some small error rates were also observed. A message retransmission can be a solution as it was mentioned in the previous section. Here we are presenting an improvement without requiring a message to be retransmitted.

Basically, the small error rates observed before were due to round-up errors. Bear it in mind that the actual clock difference is smaller than one unit of the synchronization precision (and the synchronization precision defines the maximum offset between any two clocks in a network) if the assumption on the achievable synchronization precision is correct. For example, if the synchronization precision is one millisecond, then the clock difference cannot be larger than one millisecond. The Receiver will be able to guess the correct time value used by the Sender when a round-up error happens at the Receiver. In this case, the Receiver's clock is either the same as, or maximum one precision unit larger than, or maximum one precision unit smaller than the Sender's clock.

Therefore, we propose a solution to recover from the receiving errors. The Receiver will first try to decrypt with its rounded local time. If that does not work (and a failure is detected when the received message authentication code cannot be verified), it will try to increase its rounded local time by one precision unit and try again. If the reception still fails, the Receiver will decrease its rounded local time by one precision unit and try again before a final failure can be claimed. Table II lists the results.

TABLE II. TIME SYNCHRONIZATION PRECISION VS. ERROR RATE (WITH RETRIES).

Synchronization Precision	Error Rate (Clock difference = 10s)	Error Rate (Clock difference = 1s)
1 day	0	0
0.1 day	0	0
0.01 day	0	0
0.001 day	0	0
0.0001 day	37%	0
0.00001 day	100%	100%

From Table II, it can be seen that there is no receiving error when the actual clock difference between the Sender and the Receiver is smaller than one unit of the assumed time synchronization precision. This is a significant improvement compared to the results shown in Table I. When the synchronization precision is 0.0001 day (equivalent to 8.64 seconds), the error rate is 37% if the clock difference is 10 seconds. This is because the actual clock difference (i.e. 10 seconds) is now larger than one unit (i.e. 8.64 seconds) of the assumed synchronization precision (or we say it is a wrong assumption on the synchronization precision), and would require a more aggressive adjustment of the local clocks (e.g. by adding or subtracting two precision units from the local time) in order to reduce the error rate. When the synchronization precision is 0.00001 day and the clock difference is 1 second, the result should be similar to the case when the synchronization precision is 0.0001 day and the clock difference is 10 seconds, but we have observed a very different error rate of 100% instead of 37%. We believe this is because that the computational delay (due to multiple runs of cryptographic functions) has not been considered in the clock difference (but it should) and the computational delay has become much more sensitive at the sub-second level. In this case, the actual clock differences including computational delays would be a little bit (at the sub-second level) larger than the numbers shown in Table II.

In general, the results presented here are perfect because there is no error as long as the assumption on time synchronization is correct. Also, the solution presented in this section removes the necessity of message retransmission (which is usually expensive), and only adds extra computation overhead in case of message authentication failures.

VI. CONCLUSIONS

In this paper, we have proposed a new session key agreement protocol for time synchronized networks. The new protocol removes the need of using a *nonce* (or any challenge-response method), and relies on the synchronized local time to guarantee freshness of an updated session key. In comparison to the Wi-Fi WPA2 protocol, the proposed protocol is lightweight and not vulnerable to the newly discovered Key Reinstallation attack [8]. The proposed protocol is especially suitable to be used in low-data-rate communication networks.

REFERENCES

[1] Q. Wang and I. Balasingham, "Wireless sensor networks - an introduction," in *Wireless Sensor Networks: Application-Centric Design*, T.Y. Kheng and G. V Merrett, Eds. ch. 1, InTech, 2010 , pp. 1–14.

[2] P. Lea, *Internet of Things for Architects*, Packt Publishing, 2018.

[3] Zigbee, http://www.zigbee.org/.

[4] Bluetooth, https://www.bluetooth.com/specifications

[5] Network time protocol. Available at: https://tools.ietf.org/html/rfc5905

[6] W. Diffie, M. Hellman, "New directions in cryptography", Proc. of *the AFIPS National Computer Conference*, 1976.

[7] IEEE 802.11i-2004: https://standards.ieee.org/findstds/standard/802.11i-2004.html.

[8] M. Vanhoef, F. Piessens, "Key reinstallation attacks: forcing nonce reuse in WPA2", Proc. of *the ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, pp. 1313-1328, 2017.

[9] Matlab, https://www.mathworks.com.

[10] HMAC: Keyed-hashing for message authentication. Available at: https://www.ietf.org/rfc/rfc2104.txt.

[11] W. Stallings, and L. Brown, *Computer Security – Principles and Practice*, 3rd ed., Pearson, 2015.

[12] Advanced encryption standard (AES), FIPS PUB 197, NIST, 2001. Available at: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf.

[13] N. Ahmed, H, Rahman, Md.I. Hussain, "A comparison of 802.11ah and 802.15.4 for IoT," ICT Express, Vol.2, No. 3, Sep. 2016, pp. 100-102.

[14] B. Liu, Z. Yan, C.W. Chen, "Medium access control for wireless body area networks with QoS provisioning and energy efficient design," *IEEE Trans. on Mobile Computing*, Vol. 16, No. 2, Feb. 2017, pp. 422-434.

[15] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design Principles and Practical Applications*, ch. 16, Wiley Publishing, 2010, pp. 259-268.