Journal of Engineering and Applied Sciences 13 (9): 2680-2684, 2018
ISSN: 1816-949X

The Role of Access Control and Device Authentication in the Internet of Things

Ali Shawket Thiab, Abdul Samad Bin Shibghatullah and Zeratul Izzah Mohd. Yusoh
Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka,
Durian Tunggal, Malaysia

Abstract: The new generation of wireless sensor networks that is known as the internet of things enables the direct connection of physical objects to the internet using microcontrollers. In most cases, these microcontrollers have very limited computational resources. In this study, we investigate the access control solution for the IETF standard draft constrained application protocol using the datagram transport layer security protocol for transport security. We use the centralized approach to save access control information in the framework. Since, the public key cryptography operations might be computationally too expensive for constrained devices we build our solution based on symmetric cryptography.

Key words: Internet of things, internet protocol, wireless sensor networks, transmission control protocol, constrained, Malaysia

INTRODUCTION

The term the Internet of Things (IoT) is commonly used to name a set of objects (or things) that are directly connected to the internet using the Internet Protocol (IP) stack. That is the main difference of Wireless Sensor Networks (WSN) of previous generation where nodes were organized in a local network with special protocols like ZigBee (Alliance, 2004) or Wireless HART (Chen *et al*., 2014). Connection of objects to the global network in the IoT opens the opportunity for global data analysis. Typical applications for the IoT are home automation (e.g., smart home), personal health monitoring (e.g., measurements of heart rate, pulse or temperature), building automation (e.g., control heating, electrical and ventilation systems of the building), industrial automation (e.g., control of the electrical grids) and smart cities.

Objects in the IoT (the same as in WSN) are controlled via microcontrollers that are constrained in computational power, memory space and often in power consumption. At the same time protocols that are used by general purpose computers like Transmission Control Protocol (TCP) and HyperText Transfer Protocol (HTTP) are too resource consuming to be used on highly constrained devices. Moreover, IEEE 802.15.4 (Gutierrez, 2006) radio protocol that is widely used in WSN has a limited Maximum Transmission Unit (MTU) that does not meet the MTU required by IP protocol Version 6 (IPV6). These limitations lead developers to use a special protocol stack for the IoT (Montenegro *et al*., 2007).

Devices that are connected to the IoT are highly constrained in memory and computational power. Moreover, 802.15.4 based networks are lossy have low bandwidth and high latency. In this case, maintaining access control policy information inside these devices can be difficult and evaluating access control requests may be expensive. The situation becomes even worse if the number of potential clients that have an access to a device is high and these clients are dynamically changing, hence, control policies cannot be statically preconfigured in nodes. Hence, the access control information should be managed in some server outside of the device with minimum direct communication between the server and device. At the same time, access control for the internet connected devices that utilize CoAP protocol can have 2 levels of granularity.

- Device level access when any agents that have remote access to the device have full access rights to it resources
- Resource level access when each request is supposed to be verified against access control information

As each node in IoT often perform very specific function (e.g., remote temperature sensing, control of a power switch, etc.,) resource level access control is not always necessary and device level control is often enough. CoAP specification defines the utilization of DTLS protocol for authentication and secure communication. So, DTLS can also be used for solving

Corresponding Author: Ali Shawket Thiab, Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Malaysia

the device level access control problem. If DTLS connection can be established then agent is granted full access to the device resources otherwise access is denied (Eronen and Tschofenig, 2005; Wouters *et al*., 2014).

The main goal of this study is to define the access control framework that can be built on top of the DTLS protocol running in PSK mode and evaluate this framework of the hardware platform that have limited computational resources. This research suggests the lightweight, fast and secures access control protocol that can be used on constrained devices in the IoT. The design and evaluation of the access control framework for constrained devices that is built on top of the industrial security protocols and recommendations for possible use cases of the designed access control protocol (Zhang *et al*., 2012).

MATERIALS AND METHODS

One possible solution for the problem is described by (Seitz and Selander, 2013). Researchers of this draft propose 2 dynamic access control modes for applications that are based on CoAP and DTL S protocols. Derived Key (DK) mode is based on PSK mode and provides dynamic access based on symmetric key cryptography. Authorized public key mode is based on RPK mode and involves asymmetric cryptography computations.

Both suggested modes are designed for device level access control. Since, the goal of the study is to find a solution for the problem that requires minimum resources than derived key mode is more promising and will be described in details and evaluated. In this solution, 3 types of agents are identified.

- Resource Server (RS)
- Client (C)
- Trusted Anchor (TA)

The derived key mode resembles the PSK mode with the difference that keys are not explicitly provided to the RS. Instead RS and TA share the common secret key (K_{RS}-TA) that is not known to any third party. The workflow for DK Mode is shown on Fig. 1.

In the initial step client has to request an access token from the TA. The access token consists of 2 parts: a key that will be used between the client and the RS (K_{RS-C}) that is a binary value of 128 or 256 bits and a nonce. The access token is supposed to be sent over a secure channel in spite it is not mentioned in the protocol. After receiving an access token the client can initialize a DTLS connection with RS. The connection is initialized in the PSK mode. The K_{RS-C} key is kept in secret and used as

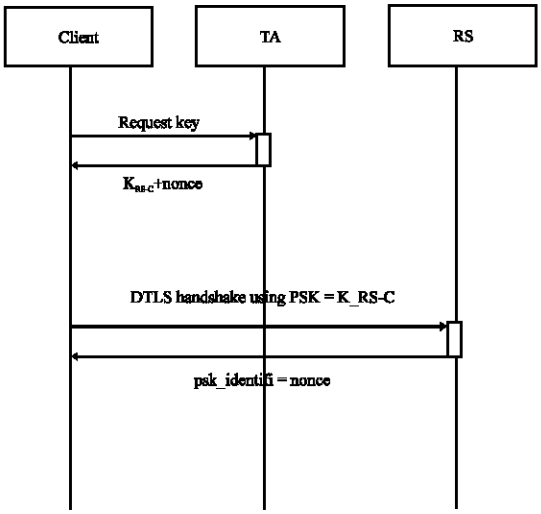

Fig. 1: Workflow for DK mode

PSK while the nonce is sent to the RS in psk_identity field of client key exchange message. Upon receiving the nonce, RS can generate K_{RS-C} from nonce and K_{RS-TA} and successfully complete a handshake.

Protocol draft (Seitz and Selander, 2013) does not define precise format of the access token. Also, authorization and authentication of the client by TA is not in the scope of the document.

Nonce contains necessary information that RS needs to authorize the connection: client identifier, identifier of TA that issued the token, sequence number of nonce per RS (this sequence number is incremented every time a new nonce is generated for certain RS). As psk_identity field in client key exchange have to be a UTF-8 string then nonce has to be sent in a string format.

- The constant nonce prefix "DK" which implies to RS that derived key mode is supposed to be used
- TA identifier that is represented in string format or converted to string format
- Client identifier that is represented in string format or converted to string format
- Sequence number that is 32 bit integer converted to string format

Both TA and RS derive the K_{RS-C} key using the nonce and the K_{RS-TA} key based on HMAC that utilizes SHA256 hash function (Krawczyk *et al*., 1997). The key is calculated using the equation K_{RS-C} = HMAC(K_{RS-TA}, nonce) where K_{RS-TA} is HMAC secret and nonce is HMAC data. HMAC produces the 256 bit value that can be used completely or truncated to the most significant 128 bits.

In order to protect the nonce from reuse and implement a key expiration authors offer a to use sliding window based on the sequence number. The RS must maintain the list of minimum 32 (recommended 64) recent sequence numbers where maximum number is the biggest sequence number that was used. Each time the RS receives the nonce it has to validate if this number is bigger than the minimum number in the list, marked as unused in sliding window or bigger than maximum number that was used. So, if the sequence number value is less than smallest value in the list than the key is considered to be expired and if the number is marked as used the RS assumes that this is a replay attack.

If TA needs to revoke the key that was not expired or terminate existing session it sends a request to RS with sequence number of a key that is supposed to be revoked. Upon request RS marks the certain sequence number in sliding window as used and terminates the active DTLS session if any.

In order to improve the protocol the nonce could be redefined in form of binary structure. The total size of the structure is 37 bytes. The first 3 bytes are the constant sequence 0×0C, 0×44, 0×4A that identifies the DK mode. The fourth byte is the identifier of the trusted anchor. Next sequence of 12 bytes is a client identifier is followed by 12 bytes of resource server identifier. Key size defines the size of the key that is supposed to be used 0 stands for 128 bits and 1 for 256 bits. The last 8 bytes of the structure is a sequence number of the key. In order to verify and evaluate the protocol 3 agents have to be implemented.

- The client is implemented in Java as a desktop application
- The Trusted Anchor (TA) is implemented in Java as a web service application
- The Resource Server (RS) is implemented in C on a constrained hardware platform

TA is running on a web server. The client is running on a desktop computer or a laptop and sends HTTP requests to the TA over a secure internet connection. The client is also connected to the Border Router (BR) over the serial port utilizing Serial Line Internet Protocol (SLIP). The RS in its turn, is connected to the BR over 802.15.4 low power radio. The client application is an application implemented in Java. It is designed to execute 3 main functions:

- Obtain the nonce and the derived key KRS-C from the TA
- Perform the DTLS handshake with the RS using granted nonce
- Send the CoAP request over the DTLS record protocol to RS

The nonce and the derived key are obtained in a JSON object from the key generation endpoint. In order to implement the DTLS handshake and the secure CoAP communication we used scandium open source DTLS implementation and Californium CoAP implementation. These frameworks were selected because they provide all necessary functionality and are easy to integrate.

RESULTS AND DISCUSSION

Evaluation: The implemented resource server part of the access control framework was evaluated on in terms of code size, memory usage, computational time and energy consumption. Since, the nonce is transmitted in the psk_identity field of the clientkeyexchange message the same as for normal PSK mode we assume zero transmission overhead for the key derivation mode. The evaluation was performed separately for key derivation and key revocation requests, since, those parts of the protocol are low coherent and can be redefined separately of each other.

In order to evaluate code size and memory usage the arm-none-eabi-size utility was used. For more detailed information about RAM and ROM usage, we used the tool arm-none-eabi-objdump. Both tools are included in the GNU toolchain for ARM processor utility. To perform a measurement for a certain functional part of the code that is in charge of specific functionality, it is removed from compilation with the #define C directive and the value is calculated as delta of the application size with and without the functional part.

The computational time is measured with the contiki energest module. This module is based on the real time clock and measures usage time of different platform units separately (CPU time, CPU time in low power mode, radio listening time and radio transmission time). The real time clock on CC2538EM platform works on 32.786 kHz frequency that allows to perform measurements with resolution 0.03 msec. The energest module accumulates the number of real time clock ticks in 64 bit unsigned values, since, its activation. That capacity is more than enough to measure the computational time of the most of access control operations. Energy consumption is calculated from time that is measured with the energest module according to the equation.

$$E = U*I*t$$

Where:

U = A power supply voltage taken from platform documentation
I = An average current for the respective module from processor specification
t = The time value measured with the energest

The energy value is calculated for each model and then summed up to take into account not only energy used by processor for computation but also potentially lost radio duty cycles.

In order to evaluate how many requests can be send by an attacker per second we measured the Round Trip Time (RTT) between client and server with a ping request. As a ping request processing time is negligible and the network consists of two nodes so the route is deterministic we assume the One Way Delay (OWD) is a half of RTT. Hence, the number of requests that can be sent by an attacker during the period of time is the length of the period divided by OWD.

The measurements were performed with ContikiMAC radio duty cycle protocol.Java client running on the PC was sending either a DTLS handshake requests sequence or a key revocation request the RS running on the embedded platform depending on the evaluated function. Data was collected 30 times for each function. The collected data were analysed to make a conclusion about performance of the access control framework and its resistance for Denial of Service (DoS) and drain battery attack that are specific for constrained devices

Access control framework evaluation: The total size of the access control framework is 1708 bytes including 1636 bytes of code, 48 bytes of static information and 24 bytes allocated for global variables. This number also includes the 16 bytes long RS identifier and 78 bytes long key KRS-TA. The part of the key revocation functionality is 392 bytes in total. The performance of the framework was measured separately for the key derivation function and for the key revocation function. Measurements results are presented in Table 1.

The key derivation with software SHA256 computation takes in average 75.47 ticks or 2.30 msec while the same computation with SHA2 hardware accelerator takes 14.6 ticks or 0.45 msec. The key revocation with and without SHA2 accelerator takes 73.7 ticks (2.25 msec) and 11.4 ticks (0.35 msec), respectively.

DTLS evaluation: In order to compare the impact of the key derivation mode and SHA2 accelerator on the DTLS handshake we measured the processing time for each message send by client. This time is measured starting from receiving the message and until the reply is sent. Moreover, the total handshake time starting from the first ClientHello message and ending with processing of the last finished message was measured to get an idea about the maximum number of handshakes that can be processed by the device within a unit of time.

Table 1: Performance measurements of the key derivation and revocation

Variables	Computation time (msec)	Energy (μJ)
Key derivation	2.30	62.79
Key derivation with SHA2 accelerator	0.45	12.29
Key revocation	2.25	61.43
Key revocation with SHA2 accelerator	0.35	9.56

Table 2: Performance measurements

Variables	Computation time (msec)	Energy (μJ)	Time (ms) with SHA2 accelerator	Energy (μJ) with SHA2 accelerator
First ClientHello	2.32	63.29	0.72	19.80
ClientHello with cookie	3.55	96.78	1.32	35.99
ClientKeyExchange	0.37	10.16	0.06	1.66
ChangeCipherSpec	17.89	488.46	3.30	90.00
Finished	9.96	271.99	2.65	72.40
Total	38.64	1054.96	8.85	241.52

The total size of the tinydtls library (including the access control framework) is 21592 bytes. This code footprint includes 19368 bytes of program, 140 bytes of constant data and 2084 bytes allocated for global variables. Results of performance measurements are presented in Table 2.

The minimum time for the complete handshake for SHA256 Software computation is 533.05 msec and average time is 775.05 msec. In case the SHA2 hardware accelerator is used the minimum and average time is 511.65 and 711.11 msec, respectively. The proposed solution in this study used the proactive WPA/WPA2 to protect the access link and IPsec security to secure the data on the internet side. A possible future research can target the availability aspect of the WPA/WPA2 access networks. While the 802.11i standard has strong measures for both confidentiality and data integrity but very little work targeted the defense against DoS attacks. Although, some intrusion detection systems or other solutions can be implemented but an integral solution that is part of the Wi-Fi standard should exist.

One way delay: The average RTT value was 362.95 msec. Hence, the average OWD value is 181.48 msec and we can assume that attacker can send 8.44 requests per second.

Analysis of the access control framework code footprint shows that key derivation and revocation functions take 7.9% of the total DTLS implementation. The key derivation function takes 6.0% of total computational time per handshake. SHA2 hardware acceleration has a significant impact on the key derivation and revocation processing time for DTLS handshake messages. The accelerator speeds up the key derivation computation 5.11 times and the key revocation computation 6.43 times. Also, the accelerator speeds up overall computation time of the handshake 4.37 times.

We found out by comparing overall handshake time with and without SHA2 hardware acceleration, that optimization of the computational time has almost no impact on total handshake time. Hence, the key derivation has no significant impact on total DTLS handshake time.

CONCLUSION

The intention of this study was to design and evaluate the access control protocol that is suitable for resource constrained devices that connect objects to the internet of things. We analyzed existing solutions and based our approach on the IETF draft that is based on the DTLS protocol in PSK mode. The draft was analyzed and a few improvements were offered before implementation and evaluation.

The protocol was implemented and evaluated on CC2538 platform that includes the low power ARM Cortex M core and 802.15.4 radio module. This hardware configuration is commonly used in IoT applications. Evaluation results shows that the access control framework increased computational effort of the DTLS handshake by 6.0%, increases the code footprint of the DTLS implementation by 7.9% and has no effect on the overall handshake time. We found out by analyzing computational time that the protocol is not vulnerable to deny of service or battery drain attack.

Moreover, the DTLS handshake protocol in PSK mode was evaluated using the CC2538 platform and the ContikiMAC radio duty cycle protocol. We found out the computational efforts required for processing handshake messages and average overall handshake time. We also, compared results with software and hardware implementation of the SHA-2 hash function and found out that the hardware accelerator speeds up message processing computations in 4.37 times but has no effect on overall handshake time. Analysis of attack vulnerability shows that with a limited number of session slots that is expected in IoT applications it is easily possible to make the device unresponsive for about 2 min.

SUGGESTIONS

According to evaluation results, we offered to add roles as the functional extension of the protocol. Moreover, we suggest improving the key derivation procedure to reduce computational efforts for processing fake or erroneous nonces and to improve the key revocation procedure to perform multiple key revocation in one request. In addition to this, we recommend to define access control protocol as a DTLS extension in order to prevent the denial of service attack mentioned above. The resulting protocol is feasible to use in the IoT. We recommend this approach for application that require dynamic centralized access allocation, reliable user authentication and authenticated encryption of data transmitted in both directions. The typical usage example can be pay-per-use applications in the IoT.

REFERENCES

Alliance, Z., 2004. ZigBee document 053474r06. Version, 1: 1-14.

Chen, D., M. Nixon, S. Han, A.K. Mok and X. Zhu, 2014. Wireless HART and IEEE 802.15.4e. Proceedings of the 2014 IEEE International Conference on Industrial technology (ICIT), February 26- March1, 2014, IEEE, Busan, South Korea, ISBN:978-1-4799-3940-4, pp: 760-765.

Eronen, P. and H. Tschofenig, 2005. Pre-shared key ciphersuites for transport layer security (TLS). Network Working Group, 1: 1-15.

Gutierrez, J., 2006. Wireless Medium Access Control and Physical Layer Specifications for Low-Rate Wireless Personal Area Networks. IEEE Press, New York, USA.,.

Krawczyk, H., R. Canetti and M. Bellare, 1997. HMAC: Keyed-hashing for message authentication. Network Working Group, 1: 1-11.

Montenegro, G., N. Kushalnagar, J. Hui and D. Culler, 2007. Transmission of IPv6 packets over IEEE 802.15.4 networks. Network Working Group, 1: 1-30.

Seitz, L. and G. Selander, 2013. Additional security modes for CoAP. IETF, Fremont, California, USA.

Wouters, P., H. Tschofenig, J. Gilmore, S. Weiler and T. Kivinen, 2014. Using raw public keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). Internet Eng. Task Force, 1: 1-18.

Zhang, R., Y. Zhang and K. Ren, 2012. Distributed privacy-preserving access control in sensor networks. IEEE. Trans. Parallel Distrib. Syst., 23: 1427-1438.