ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 9, Issue 3, pp: (4-9), Month: July - September 2021, Available at: www.researchpublish.com

An End to End Lightweight Encryption Scheme at CoAP Proxy in an IoT Environment

Swapnika Dulam[1], Ramanaiah O.B.V[2]

[1,2] Department of Computer Science and Engineering

[1,2] JNTUH College of Engineering Hyderabad, India

Abstract: **CoAP (Constrained Application Protocol) is a lightweight web services-based protocol like HTTP for IoT devices. CoAP requires proxies or gateways for the deployment of a fully end-to-end paradigm between end-clients (Web-browsers) and end-nodes (Sensors/Actuators). A proxy acts as a CoAP endpoint to perform communication on behalf of the client with the existing Internet, where translation between the two protocol schemes i.e. HTTPs and CoAPs occurs. If CoAP Proxy is compromised, various attacks are possible which can lead to loss of Confidentiality and Integrity of the IoT system. To provide the customized security to CoAP Proxy, End-to-End Encryption between CoAPs client and HTTPs server is used. To enforce access control mechanism the server stores the access details of the clients during the installation itself. In the next step the server verifies the identity of the client and exchanges its public key to continue the session. Public Key Cryptography is used for encryption and the identifiers are provided through X.509 certificates. The implementation is resistant to various attacks like replay attacks, meet-in-the-middle attack, etc. For encryption, we use Elliptic Curve Cryptography (ECC), which provides a higher-level security compared to the existing encryption techniques like RSA, AES, etc. by using shorter key length and thus, resulting in less computational overhead. A Java-based system is developed to study and evaluate the proposed mechanism. It is also verified for various claims (attacks) using a popular security protocol verification tool, namely, Scyther.**

Keywords: **CoAP, CoAP Proxy, ECC, Scyther.**

I. INTRODUCTION

The main idea of IoT is to create a network of connected devices. These entities could be computers, books, cars, home appliances, Smart phones, etc., and have a locatable and readable address on the Internet. They can communicate by opening a channel with any other entity, providing, and receiving data at any time [2]. The IoT facilitates a wide range of applications which depend on sensitive information that needs to be secure. Critical data and personal information will be collected by sensors, cameras, and other devices from different users like, companies, governments, hospitals, and other users. Thus, security remains to be one of the most sensitive issues given the danger if the system falls under attack. Because of the combination of both objects and communication networks, the IoT applications can be vulnerable in physical and cyber space [4].

Once the network layer is compromised, it is very easy for an attacker to gain control and maliciously use a device as well as attack, other devices nearby through the original compromised node. Appliances that maintain an online presence are easy to attack. These devices that do not have any virus protection or malware protection are highly susceptible to being used as “bots” to forward malicious code to infect other devices .Therefore, a major concern in adopting and implementing this new technology is security and privacy [5].

Building an interoperable and interconnected system requires the adoption of standard communication protocols. The protocol stack of these smart objects needs to match classical Internet hosts to make it feasible to create the so-called extended Internet. Hence, many of the security mechanisms already defined and currently used for the Internet can be reused in IoT scenarios.

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. It is designed to easily interface with HTTP. However ,the main challenge is that CoAP has not been completely integrated into the existing Web clients without the need of intermediate application servers, proxies or gateways for the deployment of a fully end-to-end paradigm between end-clients (e.g., Web-browsers) and end-nodes (e.g., Sensors/Actuators).

Even when CoAP follows up the REST style, and it could be a lightweight HTTP protocol, this is not directly interoperable with the current Web Services-enabled clients such as browsers. For that reason, this work pursuits the integration of CoAP-support over end-clients in a transparent and scalable way, considering the end-to-end foundation of the Internet and the integration with the existing architectures and applications [7].

Proxy plays the role of interface between client and Server. It can also decrypt the received message and encrypt data according to the used security transport protocol of the other side. The vulnerability appears during this phase, especially, where the proxy is not confident or supervised by an illegitimate entity. Consequently, passing through the proxy communication node, security services like confidentiality and integrity can easily be compromised. Existing security solution for CoAP deploys Datagram Transport Security layer (DTLS) and Transport Security layer (TLS) between client and server as loT (Internet of Things) communicating entities .Exploiting advantages of studied cryptographic algorithms, the focus here is on customized security objectives regarding proxy element and DTLS-TLS translation. Vulnerability(s) and attack(s) can occur on the proxy [1].

The rest of the paper is organized as follows. The section II explores more about the CoAP Protocol, in Section III Related Work is covered. Section IV deals with The Proposed Work, and section V the Results. Section VI is the conclusion.

II. CONSTRAINED APPLICATION PROTOCOL(COAP)

CoAP as already mentioned above is a lightweight replica of HTTP. The protocol is designed for machine- to-machine (M2M) applications such as smart energy and building automation. It provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments [11].

CoAP makes use of GET, PUT, POST, and DELETE methods in a similar manner to HTTP. However, unlike HTTP, CoAP deals with these interchanges asynchronously over a datagram-oriented transport, such as UDP, most suitable in constrained environments. Also, having UDP in transport layer, unlike HTTP, CoAP supports the use of multicast IP destination addressing, thus enabling multicast requests.

All CoAP traffic can be supported through only four types of messages:

Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK), Reset (RST)

Message reliability is provided by marking as CON, eventually retransmitting it on a default timeout basis until a corresponding ACK is received from the corresponding endpoint. However, when a message does not require reliable transmission (for example, each single measurement out of a stream of sensor data) it can be sent as NON. As CoAP is by default bound to unreliable transports such as UDP, messages may arrive out of order, appear duplicated, or go missing without notice. For this reason, CoAP implements a lightweight reliability mechanism, without trying to re-create the full feature set of a transport like TCP.

By using the one of the three DTLS modes, PreSharedKey mode, RawPublicKey mode , Certificate mode , for securing UDP transport for CoAP, the new architecture is called CoAPs (secured), just the same as HTTP secured with SSL/TLS is replaced by https [8]. In this way, the security association can be used to authenticate (within the limits of the security model) and, based on this authentication, authorize the communication partner, since CoAP itself does not provide any protocol primitives for authentication or authorization.

Using full DTLS for securing CoAP may introduce a significant overhead in constrained environments with either limited local memory on nodes, or limited bandwidth on communication, or both, affecting the overall solution efficiency.

Therefore, depending on application, DTLS modes may be configured by disabling all the unnecessary ones, thus making the protocol much lighter.

CoAP supports a limited set of HTTP functionality and thus cross-protocol proxying to HTTP is straightforward. Proxying is accomplished by means of an intermediary. However, care should be taken when designing and authorizing such solutions because of vulnerabilities that can be introduced [8].

III. RELATED WORK

HTTP proxies offer a transport-protocol proxying function ("CONNECT") to enable end-to- end transport layer security through the proxy. This HTTP CONNECT method can be used as a tunnel by client application to connect to a HTTPs Server. This method starts two-way communications with the requested resource. It is used to open a tunnel. For example, the CONNECT method can be used to access websites that use SSL (HTTPS). The client asks an HTTP Proxy server to tunnel the TCP connection to the desired destination.

This method cannot currently be satisfied by an HTTP-CoAP proxy function, as TLS to DTLS tunnelling has not yet been specified. For now, a 501 (Not Implemented) error is returned to the client [11]. When a client uses a proxy to make a request that will use a secure URI scheme (e.g., "CoAPs" or "HTTPs"), the request towards the proxy should be sent using DTLS except where equivalent lower-layer security is used for the leg between the client and the proxy. This ensures security of the information being transported between the two communication end points.

We aim at proposing such an implementation of CoAPs- HTTPs proxy which is secure, lightweight, and resistant to various attacks.

No such function is defined for CoAP-to-CoAP proxies in this specification, as forwarding of UDP packets is unlikely to be of much value in Constrained RESTful Environments. The authors of [1] propose one such implementation where Modified ID-KEM based on “Three pass protocol of Shamir” is used to secure the data at proxy. They specify use of symmetric encryption for further transaction which has been exchanged between the communication entities using the three-pass protocol.

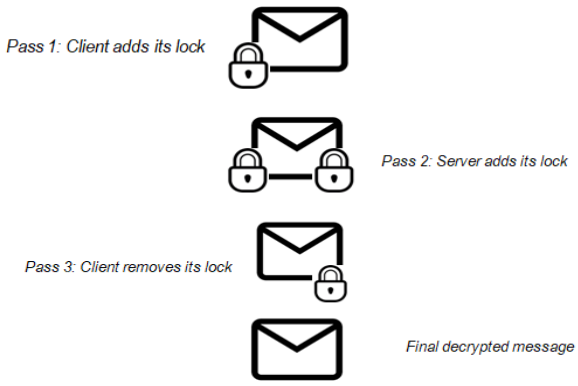

Figure 1: Shamir’s Three Pass Protocol

However, Shamir’s algorithm requires use of an algorithm that satisfies the property of being commutative, in other words Shamir’s algorithm requires use of commutative encryption. Commutative encryption is usually satisfied by block ciphers such as RC4 or XOR encryption, Pohlig-Hellman and SRA. However, they are not completely secure and are prone to various attacks.

Hence clearly implementing Shamir’s algorithm is not secure, so in order to overcome this we propose the usage of modified TLS-DTLS Translation which has inbuilt key exchange to reduce the number of flights during actual communication.

IV. PROPOSED WORK

A proxy acts as a CoAP endpoint to perform communication on behalf of the client with the existing Internet, where translation between the two protocol schemes i.e. HTTPs and CoAPs occurs. To support end-to-end communication security there is a need for proper authentication and access control throughout the network.

Thus, an Access Control mechanism is integrated within the HTTPs Server. The server stores the access details of the clients (sensor nodes) during the installation itself. In the next step the server only must evaluate and verify the identity of the client and exchange its public key to continue the session.

Public Key Cryptography is used for encryption and the identifiers are provided through X.509 certificates. An X.509 certificate contains, among other information, the public key of an entity and its common name. The implementation is resistant to various attacks like replay attacks, meet-in-the-middle attack, etc.

For encryption, we use Elliptic Curve Cryptography (ECC), which provides a higher-level security compared to the existing encryption techniques like RSA, AES, etc. by using shorter key length and thus, resulting in less computational overhead. A Java-based system is developed to study and evaluate the proposed mechanism. It is also verified for various claims (attacks) using a popular security protocol verification tool, namely, Scyther.

Once the TLS is established between CoAP proxy and a HTTPs Server , a DTLS needs to be established between CoAP client and the proxy , we are customising this step so that the key is exchanged between the communication entities before the actual communication starts.

Certificates on either sides provided by a trusted certificate authority ensure that client and server are legitimate entities communicating with each other.SSL uses certificates as de facto standard for communication and hence we to opt the usage of certificates to ensure interoperability with the existing internet infrastructure.

The First step in the protocol is ClientHello message which is sent from CoAP client to HTTPs server via proxy in this case the message is appended with a nonce to provide more security and prevent replay attack.

The HTTPs Server verifies the client hello message and sends back Hello Verify message with its public key and nonce after verifying the identity of the client with the help of certificates verified during initial TLS Step.

The CoAPs client now sends its public key encrypted with public key of server. This step is secure and resistant from attacks because, if the attacker manages to intercept the communication and try to get keys from this message, it is not possible to decrypt this message without private key of client. Moreover, knowing the public key of server is also not of any help because, decryption requires private key of server. Assuming that HTTPs device is not constrained like the CoAP client we take the chance of not encrypting public key, thus reducing overhead.

Once HTTPs Server gets the encrypted message from the CoAP client it decrypts the message using its private key. Now public key of client is available at the HTTPs server. The further communication from CoAP Client to HTTPs Server is encrypted with this public key of CoAPs client and the reverse communication is encrypted with public key of HTTPs server.

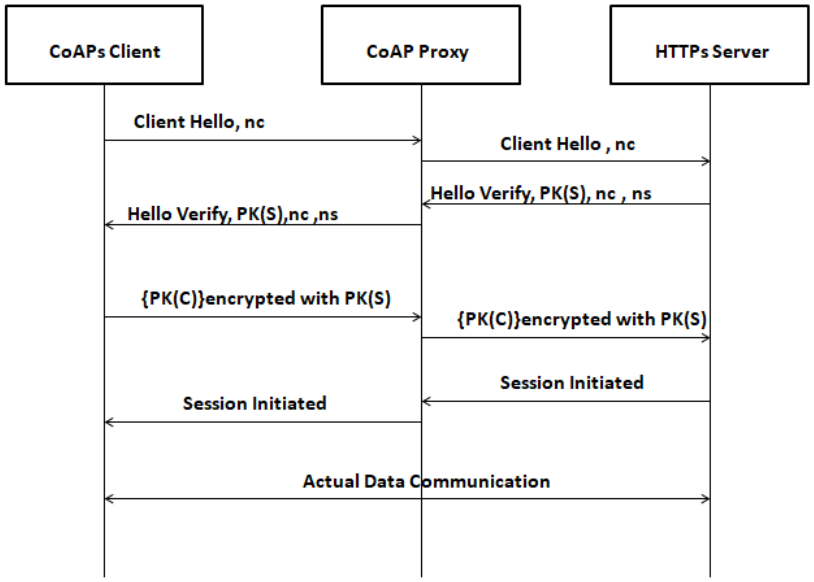

Figure 2: Proposed Message Exchange Format

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 9, Issue 3, pp: (4-9), Month: July - September 2021, Available at: www.researchpublish.com

The notations in the above Figure 2 are explained using the Table below

S. No	Symbol	Meaning
1	nc	Nonce at Client
2	ns	Nonce at Server
3	PK(C)	Public Key at Client
4	PK(S)	Public Key at Server

V. RESULT

Scyther is a tool for the formal analysis of security protocols under the perfect cryptography assumption, in which it is assumed that all cryptographic functions are perfect: the adversary learns nothing from an encrypted message unless he knows the decryption key. The tool can be used to find problems that arise from the way the protocol is constructed. This problem is undecidable in general, but in practice many protocols can be either proven correct or attacks.

Alive: is a form of authentication which aims to ensure that an intended communication party (R) has executed some events.

Nisynch: means that all received messages of R are indeed sent by the communication partner (sender) and have been received by another communication partner (receiver).

Secret: means if a term rt is claimed to be secret, rt should be kept secret to the adversary. More specifically, claim(R,secret,rt) means that R claims that rt must be unknown to an adversary.

Our Protocol is verified for various security claims to see for possibility of attacks. The following is the output generated after successful compilation of the program. The Figure shows that actual data is secret at all the three places and the protocol satisfies Alive and Ni Synch properties and that no known attacks are possible.

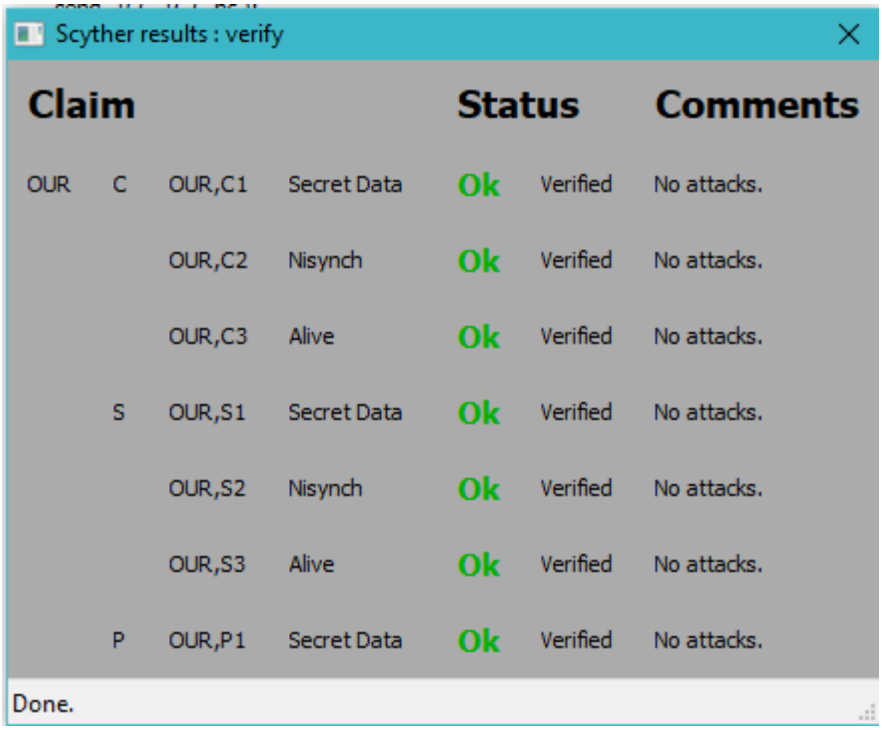

Figure 3: Scyther Output

VI. CONCLUSION

The main challenge is that CoAP is not yet completely integrated into the existing IoT system; hence it requires proxies or gateways for the deployment of a fully end-to-end paradigm between end-clients and end-nodes. A compromised CoAP proxy can be a gateway for various attacks. HTTPs to CoAPs translation has not been implemented at the moment. Our research aims to provide an End-to-End Encryption between CoAPs client and HTTPs server which ensures customized security at CoAP Proxy. An Access Control mechanism is integrated within the HTTPs Server. The client details stored on server enable an ID based authentication which is secure we use nonce to prevent Man in the middle attack and spoofing. Time stamps prevent the Replay attacks. Public Key Cryptography, that is Elliptic Curve Cryptography (ECC) is used, for encryption of the packets which provides a higher-level security by using shorter key sizes reducing the bandwidth consumption. Any protocol needs to verified if it is well formed, here Scyther, an automatic claim verification tool, is used to check if the protocol is well-formed and verifies it for various possible attacks as mentioned above. Further enhancement in terms of reducing communication overhead by using one common identity provider throughout the entire Internet infrastructure is possible (ex: OAuth2). Such an implementation eliminates the need of authentication phase. DTLS header compression mechanisms may be integrated to reduce the overhead.

REFERENCES

[1] Hajer Boujezza, Hella Kaffel-Ben Ayed , Leila Azouz Saidane, "Protection of loT Transaction Using ID-KEM Based on Three-Pass Protocol Countermeasure", IWCMC 2017, pp 423-428.

[2] Samiul Monir, “A Lightweight Attribute-Based Access Control System for IoT”, Thesis, University of Saskatchewan ,June 2017

[3] Shahid Raza, Hossein Shafagh, Kasun Hewage, René Hummen, and Thiemo Voigt,“Lithe: Lightweight Secure CoAP for the Internet of Things”,in IEEE Sensors Journal, Vol. 13, No. 10, October 2013 pp. 3711-3720.

[4] Mohammed Hassan , “Securing the Constrained Application Protocol (CoAP) for the Internet of Things (IoT)”,Thesis,Taif University, 2011.

[5] Sathish Alampalayam Kumar, Tyler Vealey , Harshit Srivastava, “Security in Internet of Things: Challenges, Solutions and Future Directions”, System Sciences (HICSS), 49th Hawaii International Conference , 2016, pp. 5772 – 5781

[6] Bob Duncan, Andreas Happe, Alfred Bratterud ,“Enterprise IoT Security and Scalability: How Unikernels can Improve the Status Quo” , Submitted. to Closer Complexis 2017, pp. 1-8, 2016.

[7] Castro M, et al. "Enabling end-to-end CoAP-based communications for the Web of Things”, Journal of Network and Computer Applications, 2014.

[8] Nastase, Lavinia. “Security in the Internet of Things: A Survey on Application Layer Protocols.” 2017 21st International Conference on Control Systems and Computer Science (CSCS) ,2017,pp 659-666.

[9] Pallavi Sethi and Smruti R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications", Journal of Electrical and Computer Engineering, Volume 2017, Article ID 9324035, 25 pages, https://doi.org/10.1155/2017/9324035

[10] Xi Chen, “Constrained Application Protocol For Internet Of Things.”, Wireless and Mobile Networking course Spring 2014,Washington University in St. Louis, available at ,https://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/index.html

[11] Z. Shelby, C. Bormann, “The Constrained Application Protocol”, RFC 7252, June 2014.

[12] Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, and T. Berners-Lee, "Hypertext Transfer Protocol HTTP/1.1", RFC 2616, https://tools.ietf.org/html/rfc2616

[13] D. McGrew, “An Interface and Algorithms for Authenticated Encryption”, RFC 5116,January 2008, https://tools.ietf.org/html/rfc5116

[14] D. Whiting, Hifn, R. Housley, N. Ferguson ,RFC 3610 – “Counter with CBC-MAC (CCM)” September 2003, https://tools.ietf.org/html/rfc3610