Trusted Lightweight Communication for IoT Systems Using Hardware Security

Arnaud Durand
Pascal Gremaud
Jacques Pasquier
arnaud.durand@unifr.ch
University of Fribourg
Fribourg, Switzerland

Urs Gerber
University of Bern
Bern, Switzerland

ABSTRACT

This paper explores cutting-edge techniques for protecting cryptographic keys in Internet of Things (IoT) systems based on web protocols. In this context, we evaluated the use of security hardware with application-layer encryption on top of the Constrained Application Protocol (CoAP) for communication between constrained devices and cloud middleware. More precisely, we propose to protect keys against tampering on devices with the help of a secure element and to use memory isolation techniques, such as those provided by Intel CPUs using Software Guard Extension (SGX), on middleware. If properly implemented, this enables privacy-preserving services where even the service provider is unable to decipher exchanged data. Finally, we validated this solution on constrained nodes by measuring performance and energy requirements on an ultra-low-power microcontroller connected to a commercial secure element.

CCS CONCEPTS

• **Security and privacy → Key management**; **Security protocols**; *Embedded systems security*; Domain-specific security and privacy architectures.

KEYWORDS

IoT, Security Protocols, CoAP, OSCORE, Hardware Security, Trusted Execution

ACM Reference Format:
Arnaud Durand, Pascal Gremaud, Jacques Pasquier, and Urs Gerber. 2019. Trusted Lightweight Communication for IoT Systems Using Hardware Security. In *9th International Conference on the Internet of Things (IoT 2019), October 22–25, 2019, Bilbao, Spain.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3365871.3365876

1 INTRODUCTION

Mass adoption of connected devices raises security and privacy issues. Security is often an afterthought when connecting constrained devices to the Internet [2]. Reconciling top-notch security with inherent limitations of constrained devices, such as limited memory or low-power requirements, is a difficult task. One particular risk of IoT systems is the exposure of cryptographic keys [7]. Network nodes may be physically accessible to attackers, letting them retrieve the private key(s) by inspecting embedded devices. Securing keys and collected data on the server end is also critical, as it is typical for IoT systems to gather a vast amount of sensitive data. Moreover, wide use of cloud-based services for IoT challenges privacy, as providers may have incentives to snoop on users' data.

Interoperability is another challenge in IoT networks; due to the large number of competing IoT standards, it is difficult to interconnect different systems. The Web of Things model [6] leverages web technologies to improve interoperability.

To solve the previously mentioned issues, we leveraged hardware security to protect the keys and data of IoT systems within web environments. This paper focuses on the scenario of constrained devices connected to IoT middleware, which can be privately-deployed or hosted on the cloud. RESTful communication with constrained devices is achieved with the Constrained Application Protocol (CoAP), as opposed to HTTP which targets more capable hardware. We secure exchanged data using Object Security for Constrained RESTful Environments (OSCORE) [12], which creates an application-layer protection, as opposed to (Datagram) Transport Layer Security ((D)TLS). This has several advantages, which are later detailed in Section 6.3.

To protect key(s) on constrained devices, we used a secure element (SE) capable of performing cryptographic operations without exposing the private key(s), and withstanding tampering and probing attacks. By using such an integrated circuit, a private key never leaves the SE and is not exposed to the application's working memory. On the resource server, we protected both the cryptographic keys and the application data in a trusted execution environment (TEE) by using Intel Software Guard Extensions (SGX) technology. An SGX enclave acts as a reverse sandbox, protecting code and data from the outside world. The proposed security scheme results in a secure lightweight web-based communication with strong isolation of keys.

We demonstrate how to use hardware security to protect IoT systems' keys and data for (1) constrained nodes and (2) cloud infrastructure within web environments. To this end, we developed a set of open-source lightweight portable libraries[1] for clients and servers that implement OSCORE and key exchange constructs. This paper describes the interaction between the components of

IoT 2019, October 22–25, 2019, Bilbao, Spain

ACM ISBN 978-1-4503-7207-7/19/10…$15.00
https://doi.org/10.1145/3365871.3365876

[1] https://github.com/TrustedThings/oscore

the system and how cryptographic keys and the application data are kept secure. We then evaluate the viability of such a solution by measuring (1) the energy and performance impact of establishing a secure context in constrained node, (2) the performance overhead of establishing a secure context in an SGX enclave running on a server and (3) the data overhead.

2 BACKGROUND

2.1 Hardware Security

Hardware-enabled security is a key element of the proposed system, since it enables key isolation and tamper resistance.

2.1.1 Secure Element. An SE is a module that safeguards digital keys and performs cryptographic operations on the behalf of an external application. In the context of this paper, the secure element is a dedicated chip, which is connected to the microcontroller of the constrained device. Cryptographic materials are completely isolated from the firmware. Secure elements are designed for tamper resistance, which typically involves a protective mesh [1]. Secure elements also provide protection against side-channel attacks.

2.1.2 Trusted Execution Environment. A Trusted Execution Environment (TEE) is a secure area in a microprocessor. Intel SGX is a set of instructions, which is the building block to create a TEE on Intel CPUs. Using this technology, it is possible to protect enclaves of code and data from the system, such as the operating system (or a hypervisor), drivers, the BIOS and firmware. These enclaves run inside a TEE, guaranteeing protection from the outside with respect to confidentiality, integrity, and freshness. This is not only useful to protect secret keys, which would typically be enforced using a trusted platform modules (TPM), but also to protect application data on remote servers. An extensive review of Intel SGX [3] has analyzed its security properties and resilience against both physical and software attacks.

2.2 Object Security for Constrained RESTful Environments

OSCORE is a standard for application-layer protection of CoAP. Messages are protected within signed and/or encrypted objects using the Concise Binary Object Representation (CBOR)—a compact alternative to the JavaScript Object Notation format. Compared to transport-layer encryption (i.e., (D)TLS), OSCORE does not obscure the messaging layer. This makes it ideal for networks with intermediate nodes, such as proxies or cross-protocol translators (e.g., mapping CoAP to HTTP). OSCORE does not provide by itself any key exchange mechanisms. We used the Ephemeral Diffie-Hellman over COSE (EDHOC) key exchange, as proposed by Selander et al. [11], a draft protocol specifically designed for OSCORE key exchange. EDHOC employs an elliptic curve Diffie-Hellman (ECDH) key exchange scheme, meaning that the keys exchanged during the protocol are points on an elliptic curve. Due to the use of ephemeral keys, it provides perfect forward secrecy.

3 SECURE CONTEXT ESTABLISHMENT

In this section, we describe our process for establishing a secure context between a constrained device and the IoT middleware. This section focuses on the procedure from a high-level perspective; the key protection mechanisms are later described for particular roles in Sections 4 and 5.

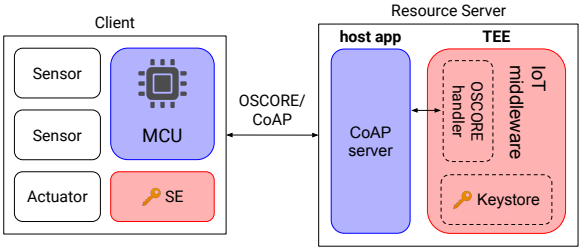

Figure 1: System overview

To clearly define the role(s) performed by each party, we will reuse the terminology from the draft *ACE-OAuth framework* [10] document. The constrained devices act as ***clients***. The IoT middleware performs both the ***resources server*** role and the ***authorization server*** role. Note that this is in contrast with some ACE-OAuth examples where the *resource server* role is performed by a constrained device.

Essentially, *clients* exchange RESTful resources with a *resource server* using the CoAP protocol. These resources are protected in CBOR Object Signing and Encryption (COSE) objects—as defined in OSCORE. Components of the system are depicted in Figure 1. In this scenario, OSCORE secrets, including those from the key exchange, are kept isolated from the external world, either using an SE or a TEE.

When establishing an OSCORE context using EDHOC, parties perform a three-way handshake: the *client* and the *resource server* mutually authenticate and derive a shared secret from an ephemeral key using EDHOC. While EDHOC provides mechanisms for authentication based on either asymmetric or symmetric keys, this paper focuses on asymmetric authentication. At the end of the key exchange, both parties derive an OSCORE secure context from the shared secret.

4 SECURE CONSTRAINED CLIENT

In our proposed scenario, the *client* is a constrained node consisting of sensors or actuators. It connects to the *resource server*, which is described in Section 5. To protect the keys tied to the OSCORE context, cryptographic operations are performed on secure hardware so that the secret keys never leak. For this scenario, we used an SE, since such modules are available as low-power designs and in small packages. Fully integrated solutions, such as secure microcontroller units (MCUs), are also able to protect application data in addition to cryptographic keys. It should be noted that only cryptographic keys are protected with an SE; application data are still stored in the system's main memory. To establish a shared secret from the key exchange and then protect OSCORE data, the SE must support all operations required by EDHOC and at least one of the ciphersuites available in OSCORE. Our experiments were based on the Microchip ATECC608A, a low-cost, low-power SE targeting IoT node security and designed to prevent extraction and cloning of secret keys.

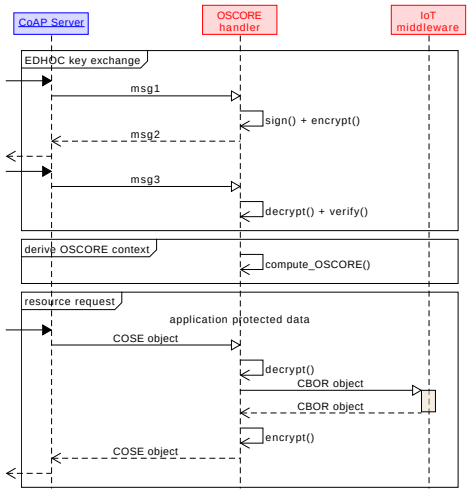

Figure 2: Sequence diagram of OSCORE context establishment as performed by the server

In order to communicate with the *resource server*, the MCU first initiates the OSCORE context establishment. This session establishment using EDHOC has a similar goal as a (D)TLS handshake: to authenticate the other end(s) and to negotiate a shared secret that will be subsequently used for symmetric encryption. During this context establishment step, the aforementioned cryptographic operations are performed on the SE. None of the client cryptographic secrets—the client private key, the ephemeral private key, and the derived AES key—ever leave the SE. The *client* secrets are stored in read-protected areas—"slots"—of the SE. Once authenticated, subsequent CoAP messages are encrypted to COSE using the previously derived AES key.

5 TRUSTED RESOURCE SERVER

In this section, we describe a secure architecture for the *resource server*, that collects and processes IoT data, using Intel SGX. We designed this *resource server* to be secure against software attacks and low-profile physical attackers. To achieve this, cryptographic operations and application-specific logic are performed inside an SGX enclave. Thanks to remote attestations, this also protects resource servers running on remote hardware. For example, as long as sensitive data stay within the enclave, code and data hosted on the cloud are kept secret from the infrastructure provider [9].

While a similar security level can be achieved with (D)TLS termination inside an enclave, using OSCORE within a TEE has several advantages. OSCORE was essentially designed to protect RESTful interactions when routed through intermediary nodes (e.g., forward proxies and cross-protocols translators). Since this kind of message processing requires (D)TLS termination, a node is also able to "*eavesdrop on, or manipulate any part of the message payload and metadata, in transit between the endpoints*" [12]. By working at the CoAP layer, OSCORE enables the system to scale to multiple enclaves, as messages can be load balanced upfront. This is necessary even for small-scale scenarios, since the working memory of a single SGX enclave is typically limited to 64MB. Another advantage is that it is possible to handle protocol operations, including CoAP handling, outside the enclave. Reducing the number of operations done within the enclave limits the attack surface due to potentially insecure code. This should also reduce the performance hit of operations running in an SGX context; this is evaluated in Section 6.2. Designing a resource server is also simpler, since it is possible to reuse existing networking libraries without restrictions. Finally, using OSCORE enables the enclave to support multiple protocols (e.g., using an HTTP-to-CoAP proxy) without modifications.

Our proposed implementation of the resource server uses a similar software stack as the constrained client. Cryptographic operations are performed by wolfCrypt, a lightweight crypto library part of wolfSSL targetting resource-constrained environments, which has specific SGX support. Both the private authentication key and the ephemeral keys are generated within the enclave (using *sgx_read_rand()* as initial seed). The authentication key can be persisted on disk using SGX *seal* operations. During the initial generation of the authentication key, it must be enforced that the authentication key was generated from within the enclave. If this step is not ensured, an attacker could pretend to run an enclave and transmit the public key of a key pair in his possession. Therefore, the enclave must be authenticated once using a remote attestation.

operation	time	energy MCU	energy SE	energy cumulated
key exchange	11.651 s	22.237 mJ	28.140 mJ	50.377 mJ
encrypt	1.352 s	2.566 mJ	3.159 mJ	5.725 mJ
decrypt	1.354 s	2.571 mJ	3.198 mJ	5.769 mJ

Table 1: Time and energy budget for OSCORE operations

6 EVALUATION

6.1 Constrained Device Requirements

To validate the applicability of the proposed solution to constrained environments, we targeted a Class 2[2] constrained node. We evaluated our secure software stack on the ultra-low-power SAML11 ARM Cortex-M23 microcontroller from Microchip Technology running at 16 Mhz. This microcontroller is connected to an ATECC608A Crypto Authentication device from the same manufacturer through an I2C bus.

To measure the transmission overhead and the energy requirement on the constrained device, we measured (1) the time to process messages and (2) the energy consumed during the session establishment. These results are presented in Table 1. Establishing an OSCORE session as the party U using EDHOC authenticated with assymetric keys takes about 12 seconds of processing time and 50 mJ of energy. While this seems long at first glance, this operation only happens once in a while (typically after a session timeout). Since energy usage is crucial for battery-powered applications, we estimated that a typical CR2032 button cell battery has enough energy to establish about 50 000 sessions or to encrypt/decrypt about 450 000 6-bytes long messages using OSCORE. Note that these figures exclude communication, which is highly dependent on the transport medium.

[2] see RFC 7228

operation	untrusted	SGX	overhead
create session	322.2 OPS	170.7 OPS	89%
encrypt	925.0 kOPS	140.5 kOPS	558%
decrypt	639.5 kOPS	132.0 kOPS	384%

Table 2: SGX performance impact on OSCORE operations. Units in operations per second (OPS).

6.2 Secure Enclave Performance

Analyses of SGX performances based on micro-benchmarks [4] show that most of the extra costs are associated with the enclave entering and exiting and the memory usage behavior. As a consequence, applications could exhibit very different behaviors when running inside an enclave.

Therefore, we evaluated this penalty (if any) for handling OSCORE messages in an enclave. As shown in Table 2, we measured the rate (on average, per second) of OSCORE operations—including EDHOC sessions establishment—our setup is able to process. Since we wanted to exclude external factors such as slow I/O (e.g., network), we built a synthetic benchmark that creates OSCORE sessions in main memory without external I/O. These tests were performed on an Intel Core i7-6700 running at a fixed frequency of 3.4 GHz (Turbo Boost and SpeedStep disabled).

These data show that there is a significant impact from establishing sessions and encrypting/decrypting OSCORE messages in an SGX enclave. To put these results into perspective, application data will be further processed by the IoT middleware in the enclave; in this situation, the application will spend on average less time in context switching between the trusted and untrusted environment, resulting in a greatly reduced performance penalty.

6.3 Protocol overhead

The overhead of both EDHOC and OSCORE has been thoroughly analyzed by Mattson et al. [8]. They concluded that "*OSCORE and DTLS 1.3 (using the minimal structure) format have very small overhead*". We reproduced these results in Figure 3 using typical IoT cloud parameters: a large connection identifier of 16-bit (as the IoT middleware is connected to many devices) with a 16-bit sequence number. During session establishment, the exchanged data is significantly reduced when using EDHOC rather than DTLS. This is shown on the right side of Figure 3, which compares the amount of data exchanged during session establishment with raw public keys. To keep the comparison fair, EDHOC is compared to DTLS 1.3 using ECDH with ephemeral keys, and the underlying CoAP message framing is taken into account. As DTLS 1.3 is work in progress and there are no available implementations at the time of writing, all DTLS results are reproduced from [8]. Since some of these results are based on work-in-progress specifications, they are bound to change.

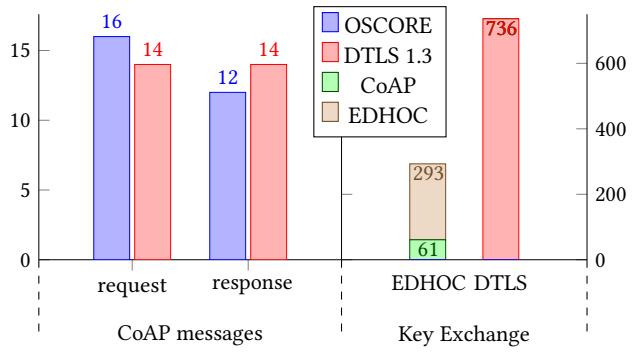

Figure 3: Data overhead. DTLS results reproduced from [8]

7 CONCLUSION

We demonstrated how to protect cryptography keys in IoT environments with web protocols using security hardware. In addition to private keys, cloud infrastructure can also protect application data, thus enabling privacy-preserving IoT middleware [5]. We also showed that application-layer encryption enables greater software flexibility, since security can be more easily decoupled from networking. Finally, we characterized the performance and protocol overhead of this setup.

REFERENCES

[1] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov. 2006. Cryptographic Processors-A Survey. *Proc. IEEE* 94, 2 (Feb 2006), 357–369. https://doi.org/10.1109/JPROC.2005.862423

[2] O. Arias, J. Wurm, K. Hoang, and Y. Jin. 2015. Privacy and Security in Internet of Things and Wearable Devices. *IEEE Transactions on Multi-Scale Computing Systems* 1, 2 (April 2015), 99–109. https://doi.org/10.1109/TMSCS.2015.2498605

[3] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.

[4] Anders T. Gjerdrum, Robert Pettersen, Håvard D. Johansen, and Dag Johansen. 2018. Performance Principles for Trusted Computing with Intel SGX. In *Cloud Computing and Service Science*, Donald Ferguson, Víctor Méndez Muñoz, Jorge Cardoso, Markus Helfert, and Claus Pahl (Eds.). Springer International Publishing, Cham, 1–18.

[5] Pascal Gremaud, Arnaud Durand, and Jacques Pasquier. 2017. A Secure, Privacy-preserving IoT Middleware Using Intel SGX. In *Proceedings of the Seventh International Conference on the Internet of Things (IoT '17)*. ACM, New York, NY, USA, Article 22, 2 pages. https://doi.org/10.1145/3131542.3140258

[6] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. 2011. *From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices.* Springer Berlin Heidelberg, Berlin, Heidelberg, 97–129. https://doi.org/10.1007/978-3-642-19157-2_5

[7] Oliver Kehret, Andreas Walz, and Axel Sikora. 2016. Integration of hardware security modules into a deeply embedded TLS stack. *International Journal of Computing* 15, 1 (2016), 22–30.

[8] John Mattsson and Francesca Palombini. 2019. *Comparison of CoAP Security Protocols.* Internet-Draft draft-ietf-lwig-security-protocol-comparison-03. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-lwig-security-protocol-comparison-03 Work in Progress.

[9] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *2015 IEEE Symposium on Security and Privacy*. 38–54. https://doi.org/10.1109/SP.2015.10

[10] Ludwig Seitz, Göran Selander, Erik Wahlstroem, Samuel Erdtman, and Hannes Tschofenig. 2019. *Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth).* Internet-Draft draft-ietf-ace-oauth-authz-22. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-ace-oauth-authz-22 Work in Progress.

[11] Göran Selander, John Mattsson, and Francesca Palombini. 2019. *Ephemeral Diffie-Hellman Over COSE (EDHOC).* Internet-Draft draft-selander-ace-cose-ecdhe-12. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-selander-ace-cose-ecdhe-12 Work in Progress.

[12] Göran Selander, John Mattsson, Francesca Palombini, and Ludwig Seitz. 2019. Object Security for Constrained RESTful Environments (OSCORE). RFC 8613. https://doi.org/10.17487/RFC8613