Lasse Jonas Rosenow

Integration of OSCORE into RIOT OS

Faculty of Engineering and Computer Science
Department Computer Science

Supervision: Prof. Dr. Thomas Schmidt
Submitted: December 24, 2023

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Contents

1 Introduction

As the Internet of Things (IoT) landscape is rapidly growing in the pasts years it becomes more relevant to ensure its robustness in terms of security. This makes it more important then ever to guarantee strong confidentiality, integrity and authenticity, that reliably works on heavily constrained devices such as they are common to find in IoT scenarios.

To make this work, standardized protocols such as Object Security for Constrained RESTful Environments (OSCORE) were created. Integrating OSCORE into RIOT OS is an important step towards a more secure IoT.

RIOT OS is an IoT operating system, that is usually running on low-tier devices with hardware constraints such as low memory, low processing power, low battery capacity. Such devices benefit a lot from a protocol such as OSCORE that was specifically designed to run on very constrained hardware.

1.1 Outline

This report provides basic background knowledge on IoT security, OSCORE [1] and RIOT OS [2], provides some related works regarding OSCORE itself and in the context of RIOT. It further defines a rough sketch on possible ways to integrate the OSCORE protocol into the RIOT operating system and concludes on which approaches may be more or less suitable.

1.2 The Problem with OSCORE in RIOT OS

RIOT OS currently does not integrate well with the OSCORE protocol. There already exists libraries such as libOSCORE[1], that can be integrated into the RIOT build process, but using them in conjunction with the RIOT Constrained Application Protocol (CoAP) stack is a tedious process. Each RIOT module or application that depends on OSCORE has to depend on a custom OSCORE implementation and can not access the OSCORE stack through a common generic interface. As a consequence the application needs to handle all the complex glue code between the OSCORE implementation and the RIOT CoAP API. This can easily cause unnecessary bugs and further increases the maintenance burden of the application itself. Furthermore having direct dependencies is problematic,

[1]https://gitlab.com/oscore/liboscore

especially for security-related libraries. If for example, a module depends on a certain OSCORE implementation, but this implementation is not maintained anymore, then the module will also not receive important security fixes anymore, until it migrates to another maintained OSCORE library. Hence wise the RIOT OSCORE integration should not be a hard dependency and should not be difficult to be updated or replaced to get new security capabilities such as updated encryption algorithms.

Having OS-level support for OSCORE, shifts this burden to the OS itself, future-proofing its maintenance for all modules within the OS that depend on it.

Ideally the RIOT OS OSCORE integration is abstracted away under the RIOT CoAP API that does not really show that OSCORE is used. Furthermore integrating OSCORE into the RIOT CoAP API should also allow changing between different OSCORE implementations such as uOSCORE[2]. So applications and modules should only use the CoAP API as they are used to do, while getting free OSCORE support by in example changing a compile time flag to enable OSCORE support and select the concrete OSCORE implementation to be used.

2 Background

2.1 IoT Security

IoT is a broad field consisting of a diverse range of devices starting from small sensors and going all the way up to large industrial machines. Letting all these device types communicate via a shared protocol is a necessity, but comes with its own challenges. One of these challenges is to harness the processing power and efficiency of large industrial machines as well as to support extremely constrained devices such as a smart sensor while still being able to guarantee strong security requirements such as confidentiality, authenticity and integrity.

Furthermore in the case of devices with extremely constrained processing powers, memory or battery life, it is necessary to find a sufficient balance between efficiency and security, guaranteeing secure operation while still being able to perform all necessary tasks that the device is supposed to do.

[2]https://github.com/eriptic/uoscore-uedhoc

Another thread to IoT security is build on the various types of communication networks that the IoT makes use of. It is not uncommon that devices communicate through multiple networks, including wireless and wired ones. This further increases the attack vector available to malicious actors.

But not only different network types but also the transmission through different protocols connected by proxies pose a thread. In some cases a large industrial machine might send a HTTP request to a wireless sensor, which only supports CoAP. To make both devices able to communicate, it is necessary to provide a proxy in-between that translates between both protocols. These proxies dramatically increase security risks as will be further explained in subsection 2.1.

To guarantee secure operation of all our IoT devices while considering all the previously mentioned challenges, it is necessary to make use of well defined and internationally standardized protocols.

One newer kind of these protocols is the OSCORE protocol.

2.2 OSCORE

OSCORE is an Internet protocol defined by the Internet Engineering Task Force (IETF). More specifically the working group that is responsible for OSCORE is called the Constrained RESTful Environments (CoRE) working group[3][4]. OSCORE is designed to address common security challenges such as authenticity, confidentiality or integrity, which are commonly faced by more constrained IoT devices such as smart light bulbs, thermostats or any other kind of edge devices.

Usually, these devices suffer from limitations such as low energy availability, low computational power, low memory and low bandwidth. These limitations make traditional security protocols such as Transport Layer Security (TLS) [3] or Datagram Transport Layer Security (DTLS) [4] a less suitable fit for constrained IoT devices. Reasons for them not being suitable are for example the need to establish sessions, which includes a heavy overhead that is especially problematic for heavily constrained devices that also happen to sporadically start or stop communications. Another limitation is the overhead

[3]https://datatracker.ietf.org/wg/core/meetings
[4]https://mailarchive.ietf.org/arch/browse/core

created by the need for re-encryption of data during a session resumption or a key renegotiation. Doing additional computation can overload the constrained processing power and will also lead to an increase of energy consumption, which in case of battery powered devices causes a shorter lifetime until the battery needs to be changed. Another limitation is that IoT devices often operate with very small packet sizes due to their bandwidth limitations. This can cause that data needs to be transferred using fragmentation, but for so small packet sizes as they can happen in the constrained IoT, the fragmentation header overhead itself can become a considerable amount of the total transferred data. As a result much more data needs to be transferred compared to the same data but without encryption.

Furthermore, communication in IoT applications often is done via the CoAP [5] protocol. But CoAP often heavily relies on the usage of proxies especially as gateways such as "CoAP-to-CoAP", "CoAP-to-HTTP" or "HTTP-to-CoAP". These proxies require TLS or DTLS to be terminated at the proxy, which discloses data to the proxy, open to manipulation. Allowing it to transform or read the payload or metadata of the CoAP message itself. Additionally, the proxy can inject, delete or reorder packets since they are not protected by TLS or DTLS. This makes proxies a security thread possibly compromising the integrity and security of our CoAP communication.

OSCORE intends to fix these IoT security issues by providing "object-level" security on top of the already existing CoAP standard. Through object-level security the transferred data itself is secured on its individual object (CoAP message) level, applying encryption, authentication and integrity protection to each CoAP message. This prevents proxies or any other middleboxes to get access or manipulate data undetected. Furthermore OSCORE is comparably lightweight. It provides security information in a compact format and comes with carefully selected, efficient encryption algorithms and provides mechanisms for mutual authentication and data integrity specified as a standardized solution, ensuring interoperability between platforms. It enables group communication, by allowing devices of the same group to establish a shared security context. It further improves efficiency by allowing to retransmit messages without the need to re-encrypt them. With the upcoming individual "Cacheable OSCORE" draft [6], it will even be possible to securely cache protected OSCORE messages at proxies.

This makes OSCORE a possibly good candidate to provide security for IoT devices that usually come with low processing power, low energy or low memory.

3 Related Work

3.1 Relevant RFCs to Improve OSCORE

The CoRE working group of the IETF is currently working on a few “active internet drafts” that propose changes or additions to the current OSCORE RFC.

Key Update for OSCORE (KUDOS) “How to update keys that the 2 peers can use to establish a new OSCORE security context?” [7]

OSCORE-capable Proxies “How to use OSCORE for protecting CoAP messages also between an origin application endpoint and an intermediary, or between two intermediaries?”, “How to secure a CoAP message by applying multiple, nested OSCORE protections?” [8]

Cacheable OSCORE “How to cache Group OSCORE messages on proxies that don’t know about OSCORE?” [6]

Protecting EST Payloads with OSCORE “How to protect EST payloads with OSCORE?” [9]

Admin Interface for the OSCORE Group Manager “How to define a RESTful admin interface for the OSCORE Group Manager?” [10]

Group Object Security for Constrained RESTful Environments (Group OSCORE) “How can OSCORE be used in a group communication setting?” [11]

Key Usage Limits for OSCORE “What are good key usage limits for OSCORE?” [12].

3.2 Academic Publications

Connecting the World of Embedded Mobiles: The RIOT Approach to Ubiquitous Networking for the IoT [13] This paper explains the design of the RIOT OS networking. It explains its "modular architecture with generic interfaces for plugging in drivers, protocols, or entire stacks", the authors further explain the "support for multiple heterogeneous interfaces and stack that can concurrently operate" and introduce the "GNRC, its cleanly layered, recursively composed default network stack".

The paper is relevant for this report, as it shows how to create a well-designed abstraction layer within RIOT OS on the example of its GNRC network stack. For the OSCORE integration into RIOT OS a well-designed abstraction layer to support plugging in different OSCORE or CoAP implementations is needed as well.

Group Communication with OSCORE: RESTful Multiparty Access to a Data-Centric Web of Things [14] This paper mainly focuses on measuring the performance of a custom "information-centric Web of Things" implementation built on CoAP and OSCORE.

While this is an interesting use case, this paper is mostly relevant in the area of our project, because the OSCORE implementation is built on RIOT OS as well. This can give us a good first hint on how to roughly start making OSCORE and CoAP work nicely together within the RIOT OS ecosystem.

Content Object Security in the Internet of Things: Challenges, Prospects, and Emerging Solutions [15] This paper compares the performances of CoAP over DTLS, OSCORE and the information-centric Named Data Networking (NDN).

The paper itself does not give insight into our project regarding the integration into RIOT OS, but further proves the advantage of OSCORE over DTLS in constrained and wireless IoT scenarios. This shows the relevance of integrating it into RIOT OS.

DNS over CoAP (DoC)," [16] This RFC defines a protocol to send DNS queries and get DNS responses via CoAP. These CoAP messages are either secured by DTLS or by OSCORE. The RFC is inspired by DNS over HTTPS, but aims the much more constrained IoT. As HTTPS has too high requirements for many IoT scenarios.

This document is relevant as it further shows an important use-case for OSCORE in IoT scenarios. Getting a more secure DNS for constrained devices without the HTTPS or DTLS overhead through CoAP with the help of OSCORE is a necessary addition to the IoT ecosystem.

Furthermore the author has published a "hacked together" OSCORE client for RIOT on GitHub[5], which can be a good guidance for implementing OSCORE into the RIOT CoAP API.

4 How to Integrate OSCORE with the RIOT CoAP API?

Since RIOT OS currently does not come with first-class OSCORE support, it proves to be relevant to research how to properly integrate it, so that first-class OSCORE support can be implemented into it. Of course OSCORE implementations such as "libOSCORE" can already be used within RIOT OS as a direct dependency, but that comes with its own difficulties and drawbacks as explained in subsection 1.2. Having OS-level support for OSCORE, shifts these difficulties to the OS itself, future-proofing its maintenance for all modules within the OS that depend on it.

To decide how to integrate OSCORE into RIOT, we need to discuss different abstraction levels. In its documentation[6] the "libOSCORE" library already specifies how to integrate it using various integration levels.

In the following paragraphs we will introduce each proposed integration level of the "libOSCORE" library.

Light Integration The light integration is the most basic way of using libOSCORE. In the light integration the application uses the CoAP API of libOSCORE directly. Furthermore libOSCORE internally uses the provided "Native CoAP API" and "Native crypto API" that are for example provided by the OS.

The advantage of this integration level is, that the application gets full control of each OSCORE step and theoretically perform optimizations for its specific use-case.

[5]https://github.com/RIOT-OS/RIOT/commit/d659ef82da025962c80e51e659b32e59c08ca469
[6]https://oscore.gitlab.io/liboscore/integration_levels

The disadvantage of this integration level is, that managing the whole encryption and decryption steps as well as the interaction between OSCORE and CoAP manually is very prone to error and requires not only a lot of work, but also a lot of inside knowledge, how to properly orchestrate everything well.

Figure 1 shows the light integration architecture.

```
                +----------------------------+
                |                            |
                |         Application        |
                |                            |
+---------------+[libOSCORE CoAP API]-+      |
|                                     |      |
|             libOSCORE               |      |
|                                     |      |
+-[nat. crypto API]-+----[native CoAP API]---+
|                   |                        |
|  Native crypto l. |  Native CoAP library   |
|                   |                        |
+-------------------+------------------------+
```

Figure 1: Light Integration Diagram

Full Integration The full integration fully hides the interaction with libOSCORE behind a CoAP API. This CoAP API internally implements all the steps of the "light integration", that otherwise would need to be implemented by the application.

The advantage of this integration level is, that applications in this case don't need to take care of the complicated OSCORE integration and just interact with a standard CoAP API. The OSCORE security context can in this case be set as a regular parameter of a CoAP message. By hiding the implementation details behind the CoAP API of the OS, it also gets much more easy for applications that previously did not provide any end-to-end security, to profit from end-to-end security without having to implement anything to make it work.

The disadvantage of this integration level is, that it removes some control from the application and thus limits the room for use-case specific optimizations.

The libOSCORE documentation considers this approach to be the most suitable way for building applications on platforms that have a powerful and stable CoAP API such as RIOT OS.

Figure 2 shows the full integration architecture.

```
+---------------------------+
|                           |
|        Application        |
|                           |
+----[original CoAP API]----+
|                           |
|    Native CoAP library    |
|                           |
+----[libOSCORE CoAP API]---+---------+
|                                     |
|             libOSCORE               |
|                                     |
+-[nat. crypto API]-+----[native CoAP API]---+
|                   |                        |
|  Native crypto l. |  Native CoAP library   |
|                   |                        |
+-------------------+------------------------+
```

Figure 2: Full Integration Diagram

Intermediate Integration The intermediate integration is a middle ground between the “light” and the “full” integrations. A “intermediate module” is provided, which simplifies the necessary steps to drive libOSCORE together with a CoAP API. The application itself interacts with a custom interface provided by the “intermediate module”. In contrast to the “full integration” this interface does not represent the “Native CoAP API” though.

The advantage of this integration level is, that it allows applications that build on top of the “intermediate module” to be portable across different CoAP APIs.

The disadvantage of this integration level is, that a new interface for interacting with CoAP is created, which does not represent the “Native CoAP API”. Thus applications can not automatically benefit from OSCORE support, but must migrate to the new API.

The libOSCORE documentation recommends this integration level if portability between different CoAP libraries is intended or if the “full integration” is not available on a platform.

Figure 3 shows the intermediate integration architecture.

```
                +----------------------------+
                |                            |
                |         Application        |
                |                            |
                +-------[bespoke API]--------+
                |                            |
                |     Intermediate module    |
                |                            |
+---------------+[libOSCORE CoAP API]-+      |
|                                     |      |
|             libOSCORE               |      |
|                                     |      |
+-[nat. crypto API]-+----[native CoAP API]---+
|                   |                        |
|  Native crypto l. |  Native CoAP library   |
|                   |                        |
+-------------------+------------------------+
```

Figure 3: Intermediate Integration Diagram

5 Conclusion

We have learned the advantages of OSCORE over traditional protocols such as DTLS in the context of IoT and why it is necessary to integrate OSCORE into RIOT (see subsection 1.2). Furthermore we found previous work that gives guidance for starting to implement OSCORE into RIOT correctly (see subsection 2.2).

In section 3.2 we learned about possible strategies in how to integrate the RIOT CoAP API with the 'libOSCORE' OSCORE implementation. From this knowledge we can conclude that a full integration should be the best fit for RIOT OS, as it was also recommended by the library itself. The full integration allows to fully hide the OSCORE implementation behind the RIOT CoAP API, allowing for seamless integration existing and future applications build on top of RIOT OS. Since RIOT OS provides a stable CoAP API called "GCoAP"[7] the risks of relying on it are negligible.

Additionally, even though the focus of this project is on integrating OSCORE support in RIOT OS, other operating systems in the IoT space such as Zephyr OS can learn from our design decisions and base their implementations on how RIOT OS solved this problem.

[7]https://doc.riot-os.org/group__net__gcoap.html

References

[1] G. Selander, J. Mattsson, F. Palombini, and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)," IETF, RFC 8613, July 2019. [Online]. Available: https://doi.org/10.17487/RFC8613

[2] E. Baccelli, C. Gündogan, O. Hahm, P. Kietzmann, M. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch, "RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4428–4440, December 2018. [Online]. Available: http://doi.org/10.1109/JIOT.2018.2815038

[3] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," IETF, RFC 8446, August 2018. [Online]. Available: https://doi.org/10.17487/RFC8446

[4] E. Rescorla, H. Tschofenig, and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," IETF, RFC 9147, April 2022. [Online]. Available: https://doi.org/10.17487/RFC9147

[5] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," IETF, RFC 7252, June 2014. [Online]. Available: https://doi.org/10.17487/RFC7252

[6] C. Amsüss and M. Tiloca, "Cacheable OSCORE," IETF, Internet-Draft – work in progress 07, July 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-amsuess-core-cachable-oscore-07

[7] R. Höglund and M. Tiloca, "Key Update for OSCORE (KUDOS)," IETF, Internet-Draft – work in progress 06, October 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-key-update-06

[8] M. Tiloca and R. Höglund, "OSCORE-capable Proxies," IETF, Internet-Draft – work in progress 07, July 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-tiloca-core-oscore-capable-proxies-07

[9] G. Selander, S. Raza, M. Furuhed, M. Vučinić, and T. Claeys, "Protecting EST Payloads with OSCORE," IETF, Internet-Draft – work in progress 03, October 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-ace-coap-est-oscore-03

[10] M. Tiloca, R. Höglund, P. V. der Stok, and F. Palombini, "Admin Interface for the OSCORE Group Manager," IETF, Internet-Draft – work in progress 10, October 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-ace-oscore-gm-admin-10

[11] M. Tiloca, G. Selander, F. Palombini, J. P. Mattsson, and J. Park, "Group Object Security for Constrained RESTful Environments (Group OSCORE)," IETF, Internet-Draft – work in progress 20, September 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-20

[12] R. Höglund and M. Tiloca, "Key Usage Limits for OSCORE," IETF, Internet-Draft – work in progress 01, July 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-key-limits-01

[13] M. Lenders, P. Kietzmann, O. Hahm, H. Petersen, C. Gündogan, E. Baccelli, K. Schleiser, T. C. Schmidt, and M. Wählisch, "Connecting the World of Embedded Mobiles: The RIOT Approach to Ubiquitous Networking for the Internet of Things," Open Archive: arXiv.org, Technical Report arXiv:1801.02833, January 2018. [Online]. Available: https://arxiv.org/abs/1801.02833

[14] C. Gündogan, C. Amsüss, T. C. Schmidt, and M. Wählisch, "Group Communication with OSCORE: RESTful Multiparty Access to a Data-Centric Web of Things," in *Proc. of the 46th IEEE Conference on Local Computer Networks (LCN)*. Piscataway, NJ, USA: IEEE Press, Oct. 2021, pp. 399–402. [Online]. Available: https://doi.org/10.1109/LCN52139.2021.9525000

[15] ——, "Content Object Security in the Internet of Things: Challenges, Prospects, and Emerging Solutions," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 19, no. 1, pp. 538–553, March 2022. [Online]. Available: https://doi.org/10.1109/TNSM.2021.3099902

[16] M. S. Lenders, C. Amsüss, C. Gündoğan, T. C. Schmidt, and M. Wählisch, "DNS over CoAP (DoC)," IETF, Internet-Draft – work in progress 05, November 2023. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-dns-over-coap-05