Lightweight X.509 Digital Certificates for the Internet of Things: Third International Conference, InterIoT 2017, and Fourth International Conference, SaSeIot 2017, Valencia, Spain...

Chapter · July 2018

DOI: 10.1007/978-3-319-93797-7_14

Lightweight X.509 Digital Certificates for the Internet of Things

Filip Forsby[1,3], Martin Furuhed[2], Panos Papadimitratos[3], and Shahid Raza[1(✉)]

[1] Security Lab, RISE SICS, Stockholm, Sweden
shahid.raza@ri.se
[2] Technology Nexus Secured Business Solutions, Stockholm, Sweden
martin.furuhed@nexusgroup.com
[3] Networked Systems Security Group, KTH, Stockholm, Sweden
{forsby,papadim}@kth.se

Abstract. X.509 is the de facto digital certificate standard used in building the Public Key Infrastructure (PKI) on the Internet. However, traditional X.509 certificates are too heavy for battery powered or energy harvesting Internet of Things (IoT) devices where it is crucial that energy consumption and memory footprints are as minimal as possible.

In this paper we propose, implement, and evaluate a lightweight digital certificate for resource-constrained IoT devices. We develop an X.509 profile for IoT including only the fields necessary for IoT devices, without compromising the certificate security. Furthermore, we also propose compression of the X.509 profiled fields using the contemporary CBOR encoding scheme. Most importantly, our solutions are compatible with the existing X.509 standard, meaning that our profiled and compressed X.509 certificates for IoT can be enrolled, verified and revoked without requiring modification in the existing X.509 standard and PKI implementations. We implement our solution in the Contiki OS and perform evaluation of our profiled and compressed certificates on a state-of-the-art IoT hardware.

Keywords: X.509 certificate · IoT · CBOR · 6LoWPAN · Contiki

1 Introduction

Most IoT standards [1,2] specify the use of digital certificates. We have recently shown that even though conventional X.509 certificates fit into state-of-the-art IoT hardware [3], they have significant overhead in terms of energy consumption on battery-powered IoT devices. Conventional certificate standards are developed for workstations and servers in mind, where factors like computational power, memory footprint and energy consumption are not main concerns. However, in battery powered and energy harvesting IoT devices, these factors are crucial and it is therefore important to adapt these standards to be more suitable for IoT. We have already adapted the Internet communication security standards

G. Fortino et al. (Eds.): InterIoT 2017/SaSeIoT 2017, LNICST 242, pp. 123–133, 2018.
https://doi.org/10.1007/978-3-319-93797-7_14

to IoT by providing the 6LoWPAN header compression mechanisms for these standards, namely IPsec [4] and DTLS [5]. In the previous work, we have either used pre-shared keys or standard X.509 certificates. There are already efforts to compress digital certificates [6] without breaking the compatibility, which uses conventional compressing methods and dictionaries with reoccurring and frequently used text strings to compress X.509 certificates. A modified version of *gzip* uses the DEFLATE [7] compression algorithm with a dictionary consisting of a typical certificate with unpopulated cryptographic fields. These solutions are designed for conventional Internet hosts; however, they can be complementarily employed along with the solutions proposed in this paper.

This paper investigates and proposes a lightweight implementation of a digital certificate with properties such as low memory footprint, low computational complexity and minimised data transfer as the main concerns. The solution proposed in this paper consists of two parts. The first part is an X.509 Profile for IoT which specify the necessary field that must be included when communicating with IoT devices, without compromising the security and standard compliance. To further reduce the size, the second part specifies compression mechanisms for the profiled X.509 certificate fields, which are applied when a certificate travels within 6LoWPAN networks. Certificates conforming to this profile will be fully valid X.509 certificates and can be processed by any entity that can process regular X.509 certificates. However, new IoT devices cannot process the legacy X.509 certificates that are generated without using the guidelines detailed in this paper. Certificates for IoT devices have to be explicitly issued using the specification of this profile. Legacy devices can get new lightweight X509 certificates that conforms to this profile.

We implement our IoT X.509 profile in Contiki, a state-of-the-art operating system for IoT. Our implementation consists of (i) traditional, (ii) profiled, and (iii) compressed X.509 certificates and their processing. We also perform empirical evaluation of our solution using a state-of-the-art IoT hardware, the ARM's Cortex M3 MCU packaged in the TI's CC2538 system on chip. Our evaluation consists of energy, memory and packet overhead, and shows significant improvements over the traditional X.509 certificates.

2 Background Technologies

2.1 X.509 Certificates

The X.509 [8] certificates has been around for a long time and are a part of many standards such as Datagram TLS [9] and IKEv2 [10]. An X.509 certificate essentially consists of three parts: *(i)* information about the subject, issuer and details about the certificate such as serial number and validity dates; *(ii)* the public key of the subject and its cryptographic algorithm; and *(iii)* a signature from the issuing Certificate Authority (CA). The latest version (X.509 version 3) has opened up for optional extensions, which can be marked as critical and thus has to be processed by the receiver. An X.509 certificate is specified and encoded

using the Abstract Syntax Notation One (ASN.1) [11] Distinguished Encoding Rules (DER), and then converted to Base64 before it is stored or transmitted.

2.2 CBOR and CDDL

Concise Binary Object Representation (CBOR) [12] is a lightweight encoding scheme with support for binary data. CBOR is designed to be extremely lightweight in terms of code and message sizes. CBOR is based on JSON [13] and fully supports the JSON syntax and data types. Even though CBOR does not rely on a specific schema in order to encode and decode messages, the CBOR Data Definition Language (CDDL) [14] was specified in order to describe and constrain CBOR structures. We use CBOR to encode and ultimately compress our profiled X.509 certificate.

2.3 IoT Protocols: 6LoWPAN, CoAP, DTLS

IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [15] is a transmission protocol that enables IPv6 communication over low-powered and lossy networks, such as the IEEE 802.15.4 protocol. The Constrained Application Protocol (CoAP) [16] is a web protocol (similar to HTTP) standardised for IoT. Secure CoAP (CoAPs) mandates Datagram TLS (DTLS) [17]. CoAP has some restrictions on how the certificates must be constructed. Section 9.1.3.3 of the CoAP standard specifies the cipher suits and subject names to be used, which we take into consideration when designing a certificate for IoT. The DTLS Profile for IoT [9] specifies the use of DTLS protocol in constrained environments. This profile also specifies the use of certificates and their contents, where restrictions have been made to keep the certificates smaller. Our work is in line with these guidelines.

3 X.509 Profile for IoT

In this section, we propose the X.509 certificate profile for IoT and discuss individual fields and their compression mechanisms. In our design, we also use the guidelines from the DTLS profile for IoT [9] standard.

Version. The current (since 2008) version is 3, which introduces optional extensions. Version 3 is also the only valid version used in the DTLS Profile for IoT. In our profiled certificate too we fixed the version value to 3. Restricting the version number allows us omitting this field while a certificate travels within 6LoWPAN networks. When a certificate leaves a 6LoWPAN network, the version field is set to 3.

Serial Number. A CA chooses the serial number during the certificate enrollment process. We do not make any restriction on the serial number value; however, we suggest low numbers and the size is reduced by encoding it in the CBOR

format. Relying on the DTLS IoT profile guidelines we also use only unsigned values.

Signature and signatureAlgorithm. These fields specify the signature algorithm that a CA uses to sign a certificate. Both the signature and signatureAlgorithm fields contain the same value. We omit both the signature and the signatureAlgorithm fields, and fix the signature algorithm to ecdsaWithSHA256, which is also used in the DTLS IoT profile. The field is populated back to ecdsaWithSHA256 by the 6LoWPAN border router when the certificate leaves 6LoWPAN networks.

Issuer. It is a non-empty sequence of name-value pairs that is used to identify the issuing CA. Though the issuer is a key field to map a given certificate to a certain CA, the range of possibilities to identify an issuer is extensive and using full range is not suitable for IoT devices. We therefore restrict this field to common name (CN) of the UTF8String type. However, the name must not be the same as for any other known CA. Our CBOR coding of this field reduces the example "Root CA" to 8 bytes from 20 bytes.

```
-- Compressed CBOR -- (8 bytes)
0x67    // Text string of size 7
0x52 0x6F 0x6F 0x74 0x20 0x43 0x41 // Value "Root CA"
```

Validity. It is a sequence of two dates: the start date and the end date, which can be represented in multiple formats. The ASN.1 representations used in conventional X.509 certificates are the longest of them all, with up to almost six times more bytes needed than UnixTime. We compress the textual format to UnixTime that requires the least amount of bytes to represent a date: four bytes before January 2038 and 5 bytes after that. Since UTCTime is implied, the structural specifiers are omitted.

Subject. Similar to the Issuer field, the subject field represents the entity with the given public key. A subject can be another CA or an end-user. In both cases it must be a non-empty Distinguished Name (DN). Relying on the DTLS IoT profile guidelines, the subject field in our profile contains the CN represented in the EUI-64 format when the certificate is issued to an IoT device. We represent the CN in the UTF8String format that is used in the IEEE Guidelines for EUI-64 [18]. Within 6LoWPAN networks, we compress the CN to the binary representation and CBOR format, which brings the size down to 9 bytes from 36 bytes. The CBOR format is represented below.

```
-- Compressed CBOR -- (9 bytes)
0x48    // Byte array of size 8
0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF  // Value 0x0123456789ABCDEF
```

Subject Public Key Info. It contains the public key in a bit string and the algorithm the key is used with. For our profiled certificate we fix the algorithm to the 256-bits ECC keys from the curve prime256v1; we therefore omit the algorithm information when a certificate travels within 6LoWPAN networks. We compress the ECC public keys using the Miller's approach [19]. Compression is done by

omitting the y-value and providing an information byte, depending on the characteristics of y. The information byte will either be 0x02 or 0x03 when the key is compressed. Since the equation of the given curve is known ($y^2 = x^3 + ax + b$), y be calculated from x as the square root of $x^3 + ax + b$. The details of compression and decompression of ECC public keys can be found in [20] Sects. 2.3.3 and 2.3.4, respectively. To further reduce the key size within 6LoWPAN networks we encode the compressed key into the CBOR format, which in total reduces its size from 91 to 35 bytes.

```
-- Compressed CBOR --  (35 bytes)
0x58 0x21    // Byte string of size 33
0x0* [ECC value X] // Where * is 2 or 3, depending on y value)
```

Issuer Unique ID and Subject Unique ID. These fields are only valid for version 2 or 3, and are only necessary if the issuer or subject are duplicated. In our solution, subjects are inherently unique, and issuers must use unique names, which makes these fields unnecessary. We therefor omit these fields in the profiled certificate.

Extensions. Extensions consist of three parts; an OID, a boolean telling if it is critical or not, and a ASN.1 DER encoded bit string as the value. We compress the OIDs by omitting the first two bytes that will always be 0x551D. The rest of the OID bytes are used as a tag for the CBOR structure which has the format: [tag, critical*, value], where `critical` is a true or false value and is the same as in ASN.1. The value will contain the DER encoded bit string, as a compression mechanism for all possible extensions and their variants will be too complex to fit in this simple protocol. Any extension is allowed in this profile. Here is an example of the compressed extension field [[1, true, 0x3000], [15, 0x03020284]].

```
-- Compressed CBOR --    (14 bytes)
0x82  /* Array of size 2 */        0x83  // Array of size 3
0x13  /* Value 1 */                0xF5  // Value true
0x42  /* Byte sting of size 2 */   0x30 0x00  // Value
0x82  /* Array of size 2 */        0x0F  // Value 15
0x44  /* Byte string of size 4 */  0x03 0x02 0x02 0x84  // Value
```

Signature. This is an encoded bit string that represents the actual digital signature of a CA. We use the ECDSA-Sig-Value format described in RFC5480 [21]. The r and s values in an ECDSA signature are both 256 bits (32 bytes) unsigned integers, when using the *prime256v1* curve. Unlike the x and y values of an ECC public key, r and s are not points on the curve and therefore cannot be compressed in the same way. There are however patented solutions for compressing ECDSA signatures, for example *Compressed ECDSA signatures* (patent number US 8631240 B2) [22], where the s value is replaced by a smaller value c. Within 6LoWPAN networks, we omit the signatureAlgorithm value (as it is fixed) and compress the signature by encoding it to the CBOR format, which reduces the size from 75 to 66 bytes.

```
-- Compressed CBOR --   (66 bytes)
0x58    /*Byte array*/ 0x40    // Size 64
[32 bytes r value] [32 bytes s value]
```

Table 1 shows the summary of all the X.509 certificate fields and their values in our profile.

Table 1. Summary of certificate field contents in the X.509 Profile for IoT.

Field	Value
Version	3
Serial number	Unsigned integer
Signature	ecdsaWithSHA256
Issuer	CommonName containing CA name as UTF8String
Validity	UTCTime in format YYMMDDhhmmssZ
Subject	CommonName containing CA name or EUI-64 as UTF8String
Subject public key info	ecPublicKey followed by prime256v1 and 64 byte uncompressed ECC public key
Issuer and subject unique ID	Not present
Extensions	Any extension
Signature algorithm	ecdsaWithSHA256
Signature	ECDSA-Sig-Value ::= SEQUENCE {r INTEGER, s INTEGER}

4 Implementation and Evaluation

We implement our proposed IoT X.509 profile in Contiki, an operating system for IoT. Our implementation supports our proposed compression, decompression, verification of compressed certificates and creation of new certificates. In our implementation, a certificate can be in three different states: Uncompressed, Compressed, and Decoded. An *uncompressed* certificate is our profiled X.509 certificate, which is used outside 6LoWPAN networks. It is represented as a byte array containing the ASN.1 DER encoded structure. A *compressed* version of a certificate is our profiled certificate compressed and encoded with the techniques specified in Sect. 3. It is used when a certificate travels within 6LoWPAN networks. It is represented as byte array containing the CBOR encoded structure. A *decoded* certificate is a C struct with all the fields from the compressed certificate. This struct is used when the certificate is verified and certain fields need to be accessed. The transitions between these stages are performed using

a number of functions. We implement the X.509 parser for encoding, decoding and working with X.509 certificates. We use two external libraries: *cn-cbor*[1] for encoding/decoding a certificate, and *micro-ecc*[2] for ECC public key creation and compression. Our Contiki app is called *xiot* (X.509 for IoT) and is placed in the `Contiki/apps/xiot` directory. All functions and types have the *xiot_* prefix. Figure 1 highlights the Contiki app.

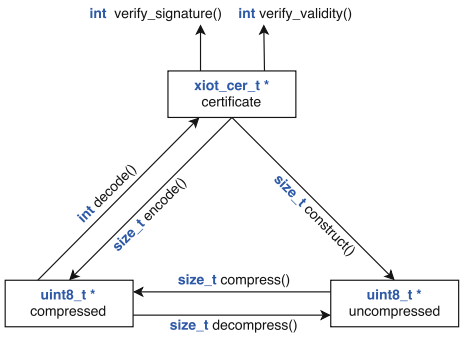

Fig. 1. Structure of our Contiki app, with functions and their interactions.

We also perform the evaluation of our profiled and compressed certificates in order to determine the overheads and gains of our solution. We measure the certificate size, per field gain, and energy consumption of our solution and compare it with the standard X.509 certificate. We perform the evaluation on the Zolertia Firefly[3] motes that uses ARM® Cortex®-M3 TI CC2538 MCU, up to 32 MHz core clock, 32 KB RAM memory, 512 KB flash memory, and power consumption from 7 mA at 16 MHz clock speed without peripherals and 20 mA or 24 mA with active radio in receive or transmit mode, respectively. It also provides hardware support for AES and ECC crypto. The full specifications can be found in the CC2538 datasheet [23] and on the Zolertia Firefly GitHub page [24].

4.1 Memory Usage

Memory overhead is evaluated in two ways: the actual size of a certificate, and the size of the compiled code. For certificate size comparison we use three different certificates: a regular X.509 certificate, a certificate conforming to the X.509 Profile for IoT, and a compressed version of the same profiled certificate. The sizes of these three certificates are shown in Fig. 2. In this case,

[1] https://github.com/cabo/cn-cbor.
[2] https://github.com/kmackay/micro-ecc.
[3] http://zolertia.io/product/hardware/firefly.

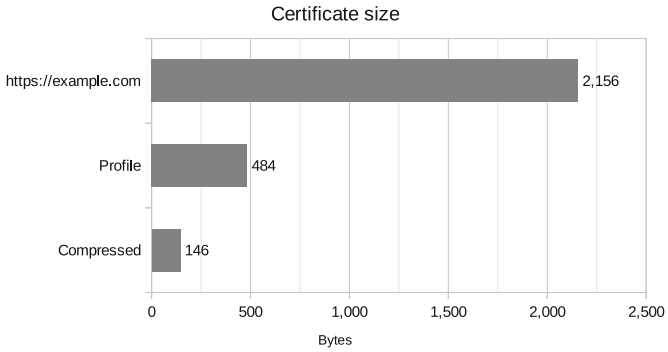

Fig. 2. The size in bytes for different example certificates.

Table 2. Size of individual fields for different certificates.

Field	Field size (Bytes)		
	No profile	Uncompressed	Compressed
Overhead	8	7	1
Version	5	5	0
Serial number	18	3	2
Signature	15	12	0
Issuer	114	20	8
Validity	32	32	11
Subject	168	36	9
Subject public key info	294	91	35
Issuer and subject unique ID	0	0	0
Extensions	596	31	14
Signature algorithm	15	12	0
Signature	261	75	66
Total	**1526**	**324**	**146**

the regular X.509 certificate is a generic example taken from the Internet web page https://www.example.com. The profiled certificate is self generated with an EUI-64 as subject and with 2 extensions. Both the profiled certificate and the example.com certificate are base64 encoded surrounded with the strings -----*BEGIN CERTIFICATE*----- and -----*END CERTIFICATE*-----, while the compressed certificate is pure binary. Base64 encoding increases the size by one third, or by ∼33%, since it takes 3 bytes and converts them to four printable characters, where each character is 1 byte. The size of a certificate can be broken down into sizes of individual fields. Table 2 shows the field sizes for the different types of certificates. These are the binary sizes without base64 encoding, and the

total sizes are therefore less than what is shown in Fig. 2. The table shows where the most bytes are used and where compression does not do much difference. It also shows for example the *subject* and *issuer* fields are greatly reduced with the profile, and so are the cryptographic parts and the extensions. The fields *version, serial number, signature, validity* and *signature algorithm* are very little affected by our profile, if any at all.

We measure the code size using the *arm-none-eabi-size* program provided for the ARM Cortex M3 platform. For the compiled code size, adding compression mechanisms on top of the regular library adds about 1.3 kB on the text area, 0.8 kB on the data area and 2 kB on the bss area.

4.2 Energy Comsumption

To measure energy, we use the Energest tool, available in the Contiki OS. Energest measures the time individual peripherals are active, and calculates power consumption using the current and voltage levels provided in the CC2538 datasheet [23]. Energest returns the time in *ticks* that must be divided by the number of ticks per second to retrieve the time in seconds. The formula for calculating energy usage is therefore: *Energy = ticks/(ticks/second) * Voltage * Current*.

Figure 3 (left) shows the energy consumption for different operations with uncompressed and compressed certificates. When no hardware support for ECC operations is used, the verification step is by far the most dominant consumer. In this case, the gain from smaller size is not as evident as when hardware crypto is used. Without hardware crypto, the uncompressed certificate consumes ∼2.2% more energy than the compressed, whereas with hardware support the uncompressed certificate consumes ∼23.4% more energy. In multi-hop 6LoWPAN networks, a digital certificate travels through multiple nodes. Figure 3 (right) shows the total energy consumption for intermediate nodes plus the end node for multiple hops. This includes decoding and verification by the end node.

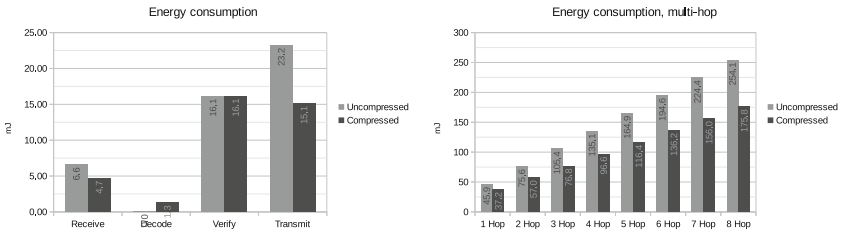

Fig. 3. *Left:* Energy consumption for different certificate handling steps, with hardware crypto. *Right:* Combined energy consumption with multiple hops.

4.3 Compatibility with the X.509 Standard

In order to prove that our profiled certificates are valid X.509 certificates, we parsed them with the well-known OpenSSL library. Our certificates pass the parsing without any errors and are X.509 compatible. For parsing an X.509 certificate with the name `certificate.crt`, the OpenSSL command we used is *openssl x509 -in certificate.crt -text -noout*.

5 Conclusion

We have specified a lightweight version of the X.509 certificate for IoT and provided compression and encoding schemes for our profiled certificate. An important feature is the compatibility with the X.509 standard, meaning that our lightweight certificate can be used in any existing PKI solutions. Our implementation for constrained environments and evaluation using a real IoT hardware show significant gains in terms of size and energy consumption.

Acknowledgement. This research is funded by VINNOVA under the Eurostars SecureIoT project.

References

1. Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). RFC 7252, June 2014. http://www.ietf.org/rfc/rfc7252.txt
2. Schaad, J.: CBOR Object Signing and Encryption (COSE). RFC 8152, July 2017
3. Raza, S., Helgason, T., Papadimitratos, P., Voigt, T.: SecureSense: end-to-end secure communication architecture for the cloud-connected internet of things. Elsevier, June 2017. https://doi.org/10.1016/j.future.2017.06.008
4. Raza, S., Duquennoy, S., Höglund, J., Roedig, U., Voigt, T.: Secure communication for the Internet of Things - a comparison of link-layer security and IPsec for 6LoWPAN. Secur. Commun. Netw. **7**(12), 2654–2668 (2014)
5. Raza, S., Shafagh, H., Hewage, K., Hummen, R., Voigt, T.: Lithe: lightweight secure CoAP for the Internet of Things. IEEE Sens. J. **13**(10), 3711–3720 (2013)
6. Pritikin, M., McGrew, D.: The Compressed X.509 Certificate Format, May 2010
7. Deutsch, P.: RFC 1951 - DEFLATE Compressed Data Format Specification version 1.3, May 1996
8. Housley, P., Ford, W., Polk, T., Solo, D.: Internet X.509 public key infrastructure certificate and CRL profile. RFC 2459, RFC Editor, January 1999. http://www.rfc-editor.org/rfc/rfc2459.txt
9. Tschofenig, H., Fossati, T.: Transport layer security (TLS)/datagram transport layer security (DTLS) profiles for the Internet of Things. RFC 7925, RFC Editor, July 2016
10. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., Kivinen, T.: Internet Key Exchange Protocol Version 2 (IKEv2). STD 79, RFC Editor, October 2014. http://www.rfc-editor.org/rfc/rfc7296.txt
11. International Telecommunication Union ITU. Introduction to ASN.1
12. Bormann, C., Hoffman, P.: RFC 7049 - concise Binary Object Representation (CBOR), October 2013

13. W3Schools. JSON Introduction
14. Vigano, C., Birkholz, H.: CBOR data definition language (CDDL): a notational convention to express CBOR data structures, September 2016
15. Kushalnagar, N., et al.: RFC 4944 - transmission of IPv6 Packets over IEEE 802.15.4 Networks, September 2007
16. Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP), March 2013
17. Rescorla, E., Modadugu, N.: RFC 6347 - Datagram Transport Layer Security Version 1.2, January 2012
18. Lambert, K.A.: Guidelines for 64-bit Global Identifier (EUI-64), January 2015
19. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_31
20. Brown, D.R.L.: Standards for Efficient Cryptography 1 (SEC 1), May 2009
21. Turner, S., Yiu, K., Brown, D.R.L., Housley, R., Polk, T.: RFC 5480 - Elliptic Curve Cryptography Subject Public Key Information, March 2009
22. Vanstone, S.A.: Compressed ECDSA signatures, November 2007
23. Texas Instruments. CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6lowpan, and ZigBee© Applications, December 2012
24. Zolertia, S.L.: Firefly - Zolertia/Resources Wiki, January 2017. https://github.com/Zolertia/Resources/wiki/Firefly