Automated Nebula Mesh VPN Certificates and Configuration files distribution

---

Gian Marco De Cola
Matr. num: 0000977684

INDEX

1 USE CASE

A Client who wants to establish a Nebula Mesh VPN [1] has no officially provided way to automatically provision CA signed Nebula certificates and configuration files to the future end nodes of the network. Instead, he/she should rely on manual deployment or secure copy (scp) of such files, implying human intervention is mandatory. The hereby proposed system wants to achieve this distribution in an automated and secure way, insuring an authenticated, tamper-proof and confidential exchange of these crucial files in order to ensure that only the authorized hosts will join the secure Nebula network, using the intended configurations. This system could help set-up a Nebula Mesh VPN in an Industrial Control System (or IIoT) settings, i.e., deploying the Nebula certs. and configs. on PLCs (programmable logic controllers), SBC (Single Board Computers, i.e., Raspberry Pis) and EWS (Engineering Workstations), to create secure, peer to peer networks between these crucial assets, that will communicate over secure and isolated channels instead of the probably unsecure and unsegregated OT network. It can also help enforce Zero Trust principles by leveraging the identity-based routing and communication infrastructure provided by the Nebula Mesh VPN.

2 SYSTEM REQUIREMENTS ANALYSIS

Requirements starting with F are hereafter deemed Functional Requirements, whereas those starting with NF are deemed Non-Functional ones. The F/NFn (i.e., F1) acronyms will be used to reference the corresponding requirements throughout the document when needed

2.1 FUNCTIONAL REQUIREMENTS

F1. The system should be able to automatically generate Nebula configuration files for all the future end nodes of the mesh network from one single configuration file
F2. The system should be able to distribute the generated nebula configuration files to the right clients
F3. The system should be able to accept requests to sign client-generated public Nebula keys
F4. The system should be able to generate Nebula private/public key pairs on behalf of low computational power clients
F5. The system should be able to create and sign Nebula certificates containing clients' Nebula public keys
F6. The system should be able to distribute the signed Nebula certificates to the right clients

2.2 NON-FUNCTIONAL REQUIREMENTS

NF1. The client and the system should be mutually authenticated to avoid unauthorized clients to gain access to the secure Nebula Mesh Network
NF2. The communication between the client and the system should be encrypted to ensure confidentiality of the exchanged messages, since they contain crucial information (i.e., encryption keys and identity certificates)
NF3. The system should be scalable and reliable enough to always be active to answer to new client enrollment requests

2.3 THIRD PARTY SOFTWARE

Fulfilling F1 should be easy enough thanks to the already provided dhall-nebula tool, which leverages the Dhall configuration language to generate multiple Nebula configuration files from one Dhall configuration file, which can be considered as an input of the system.
The Nebula source code [2] is released as open software, so it is possible to utilize its utilities to generate Nebula key pairs, certificates and to sign them as per F4 and F5.

3 PROBLEM ANALYSIS

The presented use case describes a problem of certificates and configuration files creation and distribution. The two problems will be assessed separately; in particular, the first problem seems solvable putting up in place a simple Public Key Infrastructure (PKI).

3.1 PKI CONSIDERATIONS

First of all, knowing that Nebula certificates are not X-509-compliant [3], a custom PKI solution must be put in place, as all the existing solutions are developed to manipulate X-509 certificates. This solution will concentrate on accommodating Nebula Certificate Signing Requests (as per F3) as well as Nebula key pairs generation (as per F4) and will not, for now, focus on problems such as Nebula Certificates revocation or Reenrollment requests.
Given that, a choice between developing a Public PKI rather than a Private PKI must be made.

3.1.1 Public vs Private PKI

As stated in the "Private Versus Public PKI" section of [4], for enabling a Zero Trust network, it is always recommended to use a private PKI. This is largely due to the fact that there will be an intrinsic and significant problem of trust utilizing a public PKI in a Zero Trust environment, because one must trust the CA chain of the company offering the service and, implicitly the state the company resides in. Going with a private PKI solution means managing the whole system "in house", thus trusting only what we want to trust and how we want to trust them; moreover, we also have maximum flexibility regarding how certificates are formed (which is useful managing non-standard certificates) and are managed in the system.
In both cases, it is best practice to consider the root CA of the system always offline, to protect it from external threats: this could be achieved by segmenting the system network in order to offer only one service facing the outside, while the CA remains an internal service.

3.1.2 The Enrollment over Secure Transport (EST) standard

Given our requirements (especially NF1 and NF2), some form of secure channel must be established between the clients and the PKI service. The Enrollment over Secure Transport standard [5] is an Internet Request for Comments that describes a simple certificate management protocol (based on the Certificate Management Protocol over CMS [6]) conducted over a secure transport layer in a fully automated fashion. It uses HTTPS [7] as a secure application layer protocol and a server-side REST-based APIs to invoke its functionalities, but a more recent draft also incorporates its usage over CoAPS [8] for more constrained IoT/IIoT clients.
A quick overview of the protocol will show that such standard includes all the functionalities we need for our PKI implementation and more (Table 1).

Table 1 – EST (RFC7030) Operations and corresponding URL endpoints

OPERATION	OPERATION PATH	DETAILS
Distribution of CA Certificates (MUST)	`/cacerts`	Request a copy of the current CA certificates. This function is generally performed before other EST functions.
Enrollment of Clients (MUST)	`/simpleenroll`	EST clients request a certificate from the EST server with an HTTPS POST using this operation path
Re-enrollment of Clients (MUST)	`/simplereenroll`	EST clients request a renew/rekey of existing certificates with an HTTPS POST using this operation path
Full CMC (Optional)	`/fullcmc`	EST clients request a certificate from the EST server using all the fields required in a CMC request as per its RFC, with an HTTPS POST using this operation path
Server-Side Key Generation (Optional)	`/serverkeygen`	An EST client may request a private key and associated certificate from an EST server using an HTTPS POST at this operation path
CSR Attributes (Optional)	`/csrattrs`	CA policy may allow inclusion of client-provided attributes in certificates that it issues. In addition, a CA may desire to certify a certain type of public key and a client may not have a priori knowledge of that fact. Therefore, clients SHOULD request a list of expected attributes that are required in an enrollment request at this operation path

Every client request for enrollment (Certificate Signing Request or CSR) should follow the PKCS #10 [9] or CMRF [10] formats.

Unfortunately, this standard is specified for the enrollment of X.509 certificates, and every existing implementation follows that, but its adaptation to work with Nebula Certificates should be easy enough.

3.2 CONFIGURATION SERVICE CONSIDERATIONS

The configuration service should leverage the previously mentioned dhall-nebula tool to generate nebula configurations for each client of the future Nebula network. To do so, it should receive a Dhall configuration file in its environment to load the dhall configurations and convert them into Nebula YAML config files. To know which end node configuration file to generate, the client should provide the service of the Nebula hostname it will be identified with in the future

network. If such name is not present in the list of hostnames in the Dhall configuration file, the client request should be blocked and ignored.

3.3 SECURITY CONSIDERATIONS

Having introduced the EST protocol, we have understood that its adaptation to our system would need an implementation of the TLS [11] Secure transport layer (both for the HTTPS and CoAPS cases). Other than that, we know from NF1 that both client and service must be authenticated before commencing a Nebula CSR. Of course, the TLS protocol is well capable of mutually authenticate clients and service providers and can do so in different ways, for example:

- Employing both client and service provider certificates
- Employing a pre-shared key (PSK) between the client and the service provider

The first option does not seem suitable to our case, as a client certificate should already be installed on the client node, and this means that another CA should be employed, with all the management issues that come along with it. The second option, however, will provide the same level of authentication, without the need of further infrastructure. The downside of this approach is, of course, having to deploy the PSK in a secure way (possibly out of band), and store it securely on the nodes; moreover, on the service provider side, a list of all the client PSK should be stored securely in a database for easy and fast access.
To avoid all the deployment and managing efforts linked to PSK authentication, another option is possible: one can provision a different Hash-based Message Authentication Code (HMAC) [12] of some client-identifying information to each client at deployment time, and store one single key on the service provider to validate it upon arrival (it should be sent as the first message of the client-service exchange). This approach could however lead to replay attacks of the authentication message, so this issue should be considered in the implementation phase.

A choice between these methods is delegated to the implementation phase.

Finally, one should discuss the cyphers to be employed by the TLS protocol. In doing such a choice, we should consider that clients of a Nebula Mesh Network must implement either the `CHACHA20-POLY1305` or the `AES-256-GCM` AEAD algorithms and the Elliptic Curve Diffie Hellman key exchange protocol, so the most obvious choice (for compatibility reasons) is to use:

- PSK authenticated case:
    - `TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256`
    - `TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384`
- HMAC authenticated case:
    - `TLS_ECDH_ANON_WITH_AES_256_CBC_SHA`

It should be also noted that, the only cipher suite available in the HMAC authenticated case does not ensure Perfect Forward Secrecy (PFS), as the Diffie-Hellman exchange does not use

ephemeral keys, thus reducing the security of the created channel. The cipher is also considered insecure by the current NIST cybersecurity best practices [13] and IETF drafts [14], even though a local usage could mitigate the severity of the vulnerability of such cipher.

Lastly, the proposed ciphers are TLS 1.2 compatible; the first two are supported by TLS 1.3 since they include ephemeral keys in the key exchange and use strong, state-of-the-art AEAD algorithms, while the last one is deprecated because not only uses static DH keys, but also block cipher mode of the AES algorithm and SHA-1, which are considered obsolete at the time of this writing [11].

3.4 Proposed System Architecture

Given the aforementioned considerations, every service identified in the system should be implemented as a microservice, to enhance their scalability and reliability, given that they could be easily deployed in high availability clusters (thus also fulfilling NF3). Considering this, a possible architectural solution is presented in Figure 1.

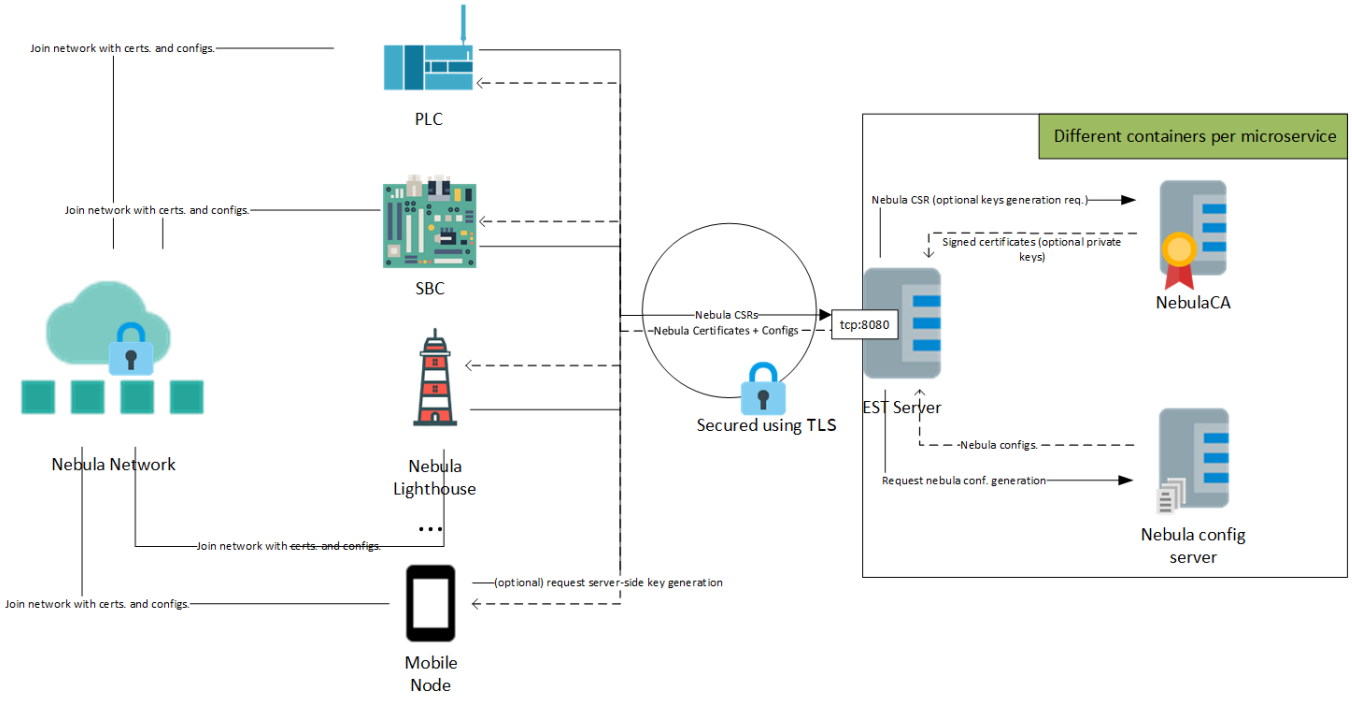

Figure 1 – Logical Architecture of the proposed system

To drive the implementation phase, high-level references of the clients-to-microservices exchange are presented in Figure 2 and Figure 3.

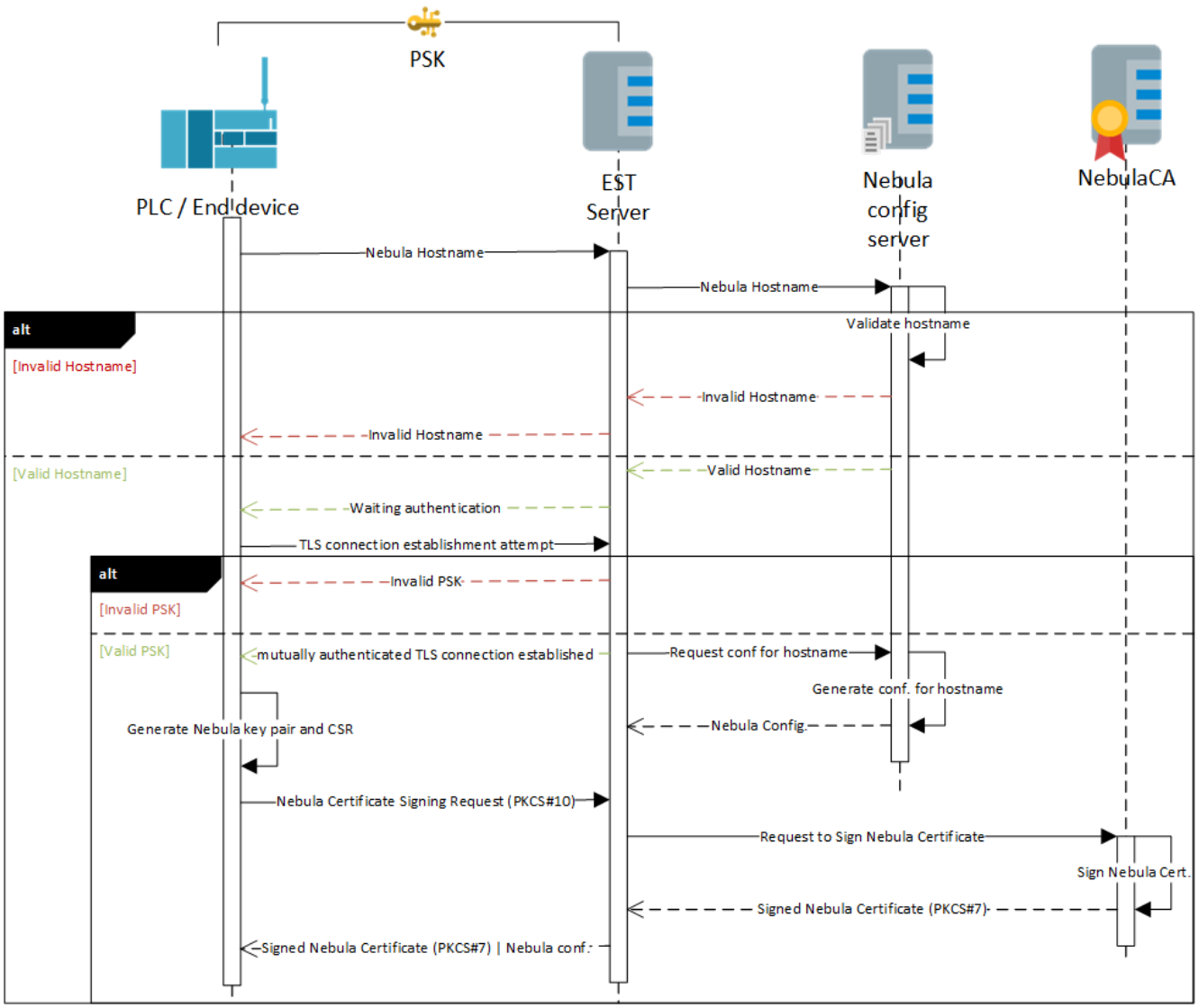

Figure 2 - High level client-services interaction (PSK case)

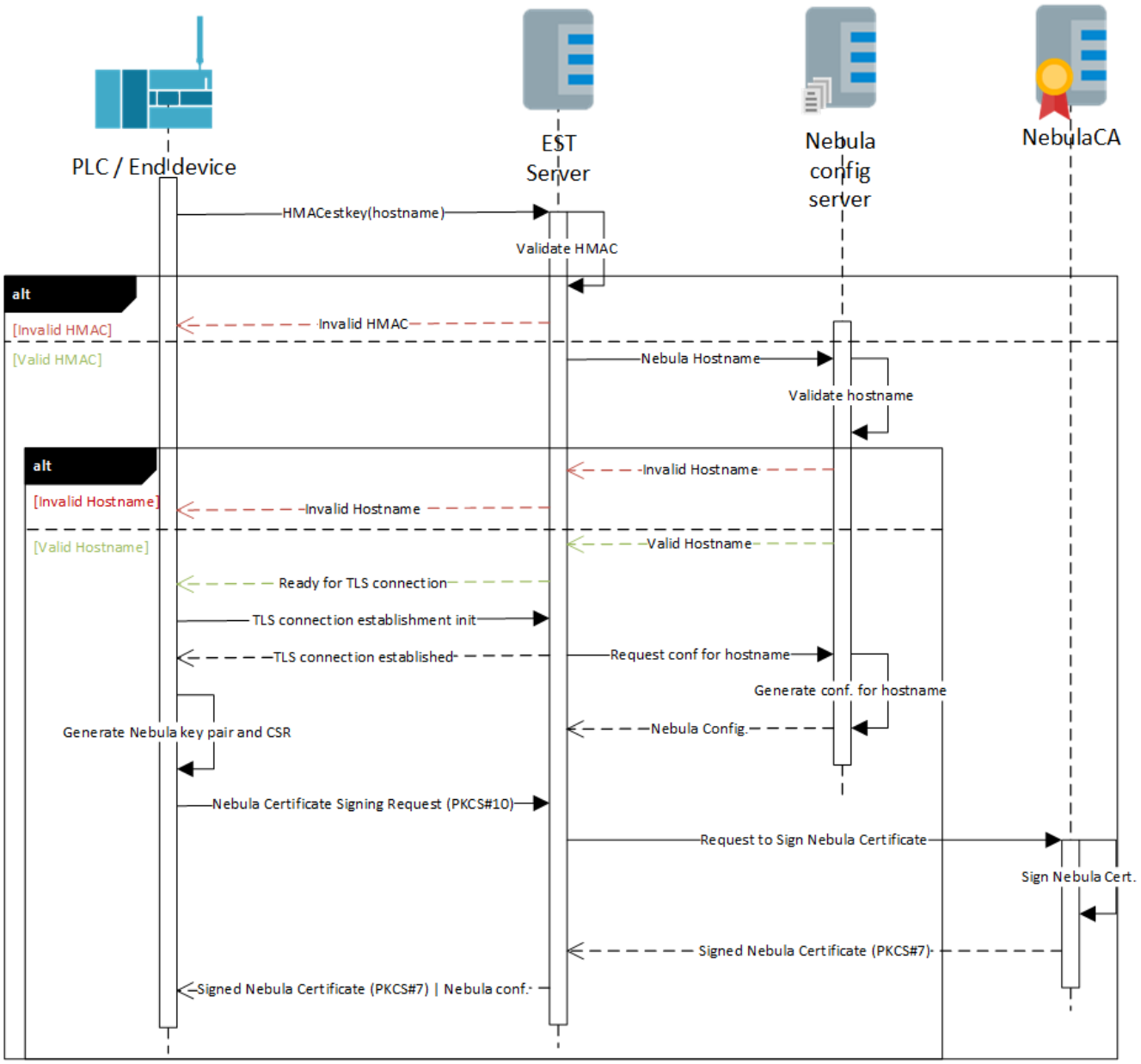

Figure 3 - High level client-services interaction (HMAC case)

Bibliography

[1] Defined Networking, «Nebula: Open Source Overlay Networking,» [Online]. Available: https://nebula.defined.net/docs/. [Consultato il giorno 11 November 2022].

[2] Slack, Inc., Defined Networking, «slackhq/nebula: A scalable overlay networking tool with a focus on performance, simplicity and security,» [Online]. Available: https://github.com/slackhq/nebula. [Consultato il giorno 11 November 2022].

[3] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley e W. Polk, «Internet X.509 Public Key Infrastructure Certificate (RFC5280),» RFC Editor, may 2008.

[4] E. Gilman e D. Barth, Zero Trust Networks, O'Reilly Media, Inc., July 2017.

[5] M. Pritikin, P. Yee e D. Harkins, «Enrollment over Secure Transport (RFC7030),» RFC Editor, October 2013.

[6] J. Schaad e M. Myers, «Certificate Management over CMS (CMC, RFC5272),» RFC Editor, June 2008.

[7] E. Rescorla, «HTTP Over TLS (RFC2818),» RFC Editor, May 2000.

[8] P. van der Stok, P. Kampanakis, M. Richardson e S. Raza, «EST-coaps: Enrollment over Secure Transport with the Secure Constrained Application Protocol (RFC9148),» RFC Editor, April 2022.

[9] M. Nystrom e B. Kaliski, «PKCS #10: Certification Request Syntax Specification (RFC 2986),» RFC Editor, November 2000.

[10] J. Schaad, «Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF, RFC4211),» RFC Editor, September 2005.

[11] E. Rescorla, «The Transport Layer Security (TLS) Protocol Version 1.3 (RFC8446),» RFC Editor, August 2018.

[12] H. Krawczyk, M. Bellare e R. Canetti, «HMAC: Keyed-Hashing for Message Authentication (RFC2104),» RFC Editor, February 1997.

[13] K. A. McKay e D. A. Cooper, «Guidelines for the Selection, Configuration, and Use of TransportLayer Security (TLS) Implementations,» National Institute of Standards and

Technology (NIST), August 2019.

[14] C. Bartle e N. Aviram, «Deprecating Obsolete Key Exchange Methods in TLS,» June 2022.