Towards a More Flexible IoT SAFE Implementation

authors
mail addresses
organization A
organization B

***Abstract*—The Internet of Things (IoT) is disseminating in everyone's daily life and gets ubiquitous not only in industry. With this growth, device and communications security is increasingly important. Hardware Security Modules (HSMs) are integrated into IoT devices to provide a "Root of Trust", and protect confidential key material used for device authentication. Due to lack of standardized interfaces, HSM manufacturers implement their own proprietary interfaces. To ease integration of hardware security, and enable vendor interoperability, the GSMA proposes IoT SAFE, a standardized interface.**

In this work, IoT SAFE is evaluated and compared against the interfaces of proprietary HSMs. Improvements are proposed in order to reduce complexity, increase flexibility, and ease the integration into Transport Layer Security (TLS) libraries. The evaluation shows that the TLS handshake performance can be improved significantly for ECC and RSA certificate-based client authentication. The message count between HSM and hosting device is reduced by approximately 40% and 25%, respectively.

***Index Terms*—IoT, TLS, DTLS, Protected Communication, eSIM, eUICC, Java Card, Applet, Interface Design, Services**

I. Introduction

The number of connected IoT devices is continuously growing since many years, and in future even stronger growth is predicted. As analyzed in [1], the number of connected IoT devices is surpassing the number of connected non-IoT devices. In 2025, this number is going to exceed 30 billion devices [1]. The more devices are interconnected, the more important it is to protect the communication links. State-of-the-art security protocols are TLS and Datagram Transport Layer Security (DTLS). Their goal to provide confidentiality, authenticity, and integrity is identical, but they rely on different transport protocols, Transport Control Protocol (TCP) and User Datagram Protocol (UDP) respectively. In order to protect important key material required during certain steps of the security protocols, the security layer shall be supported with an HSM (interchangeably used: Secure Element (SE)), providing a 'Root of Trust'. Cryptographic functions are delegated to the HSM, resulting in a partitioning of TLS functionality between hosting device and HSM. A typical TLS system partitioning is described in [2]. With this measure, the attack vectors against the communication link as well as side channel attacks are minimized [3].

Besides proprietary HSMs, the GSM Association (GSMA) defines a standardized interface, called IoT SIM Applet For Secure End-to-End Communication (SAFE), to be implemented as a Java Card applet. Fig. 1 depicts the essential architecture with the component's logical connections (dashed lines). During the (D)TLS handshake, protected key material is utilized, which must be deployed beforehand. This process is executed in advance or remotely Over-the-Air (OTA). Classical HSMs need to be provisioned during manufacturing, resulting in lack of flexibility. Embedded Universal Integrated Circuit Cards (eUICCs) are capable of being provisioned with credentials over the air, as depicted in Fig. 1. This allows "late binding" of devices, a mechanism described in [4]. It enables vendors to manufacture devices in high volume and target a customer platform at the time of installation, instead of time of manufacturing.

The initial purpose of eUICCs was to authenticate a device against a Mobile Network Operator (MNO) in order to get access to the cellular network, commonly known as Subscriber Identity Module (SIM). Due to the modular approach of utilizing applets for various services, an eUICC achieves high flexibility. Therefore, the eUICC is a multi-purpose computing platform, mostly based on Java Card.

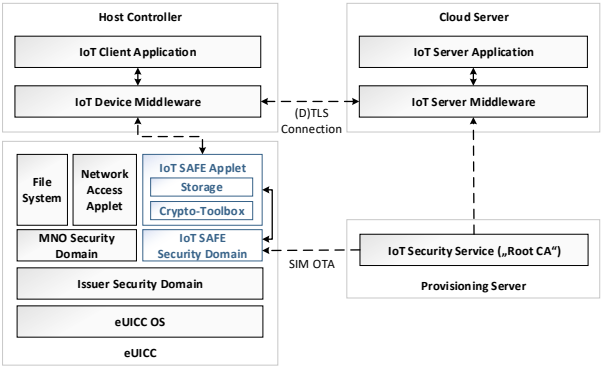

Fig. 1. IoT SAFE Concept and eUICC Architecture including Interaction with Host Controller and respective Servers (modified from [5])

Besides remote provisioning, cryptographic features, and a protected storage, the eUICC is standardized by the GSMA in [6] and inter-operable across different vendors [5]. This makes eUICCs highly qualified for being used in the IoT domain, where the market is scattered among countless manufacturers.

Not only the eUICC and the related interface is standardized, but also the applet responsible for protecting the key material and performing security critical operations during the (D)TLS handshake [7]. This applet communicates with the IoT security service to OTA provision the applet store content with key material, as depicted in Fig. 1. The IoT middleware running on the host controller, is establishing the (D)TLS

connection, supported by a cryptographic library within the IoT SAFE applet. On top of the middleware, the TLS library and the application logic is executed on the host controller.

This work focuses on the interface between applet and host controller, including the respective software components. The main contributions are:

- Structured analysis of the current version of the IoT SAFE applet
- Implementation and specification enhancement regarding complexity, simplicity, and endurance
- Evaluation of current state, improvements, and limitations

II. State-of-the-Art

For supporting various IoT applications with regard to security, many different devices are available on the market. In this work, several proprietary HSMs and the applet-based solution are evaluated and compared.

A. Proprietary HSMs for Supporting the Security Layer

In this chapter the following comparable devices and their host interface are analyzed.

- Microchip ATECC608B [8]
- NXP EdgeLock™ SE050 [9]
- Infineon OPTIGA™ Trust X [10]

Comparing these HSMs, they all have specific design characteristics. Microchip's ATECC608B is designed for simplicity and ease of integration, by offering a minimum command set required to secure typical IoT applications. In contrast, NXP's EdgeLock™ SE050, offers a complex command set, including several variations for hashing and key derivation. Further, it supports numerous signature algorithms and Elliptic Curve Cryptography (ECC) curves. The command set provided by Infineon's OPTIGA™ Trust X is more complex than Microchip's ATECC608B, but simpler than NXP's EdgeLock™ SE050.

B. IoT SAFE Applet

An eUICC offers two different connectivity options to the host controller: indirect connection via a modem, based on the ISO7816 T=0 protocol; direct connection such as I^2C or SPI. Typically, eUICCs are connected indirectly via the modem with corresponding AT and SIM commands. The T=0 protocol does not support chaining, therefore chaining must be implemented on Application Protocol Data Unit (APDU) level if required. The chosen accessibility option does not influence the implementation of the IoT SAFE applet.

In [11], a typical IoT device system architecture, and a structured evaluation of various I^2C communication protocol stacks, are described. The evaluated Global Platform APDU transfer over I^2C is a derivative of the ISO7816 T=1 protocol and is called T=1' [11]. The tremendous difference of various protocol stack implementations regarding code size, which is an important factor due to resource-constrained devices, is evaluated in [11].

The main purpose of the IoT SAFE applet is supporting the host controller during the (D)TLS connection establishment to the server, and providing a protected applet specific storage for important key material, as depicted in Fig. 1. The handshake sequence with dissected critical steps is presented in Fig. 2. In this work we focus on TLS, but the DTLS handshake engine is equivalent regarding the utilized cryptographic operations. The interfaces to access the functionality of an HSM need to be designed in a way to ensure simple integration into various TLS libraries, such as *OpenSSL* or *mbedTLS*, in order to be adaptable and reduce errors during integration.

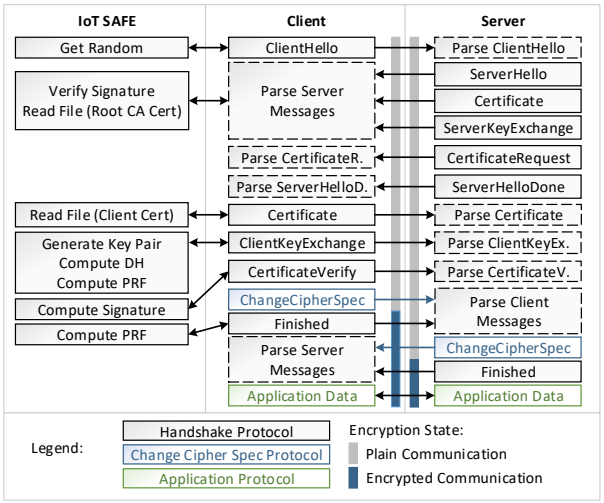

Fig. 2. TLS 1.2 Handshake with IoT SAFE (modified from [12])

The GSMA defines two different IoT SAFE applets due to resource constraints of specific IoT devices, but this work focuses on *IoT Security Applet Type 1*, since the functionality of *IoT Security Applet Type 2* is a subset. An overview of the applets functionality is depicted in [5].

The applet as well as the applet store content are provisioned OTA, by a provisioning server, which requires the key material to authenticate against the respective security domain of the eUICC, as depicted in Fig. 1. An applet can be installed in two possible places, either within an MNO profile, or in a separate security domain. The advantage of a dedicated security domain for the IoT SAFE applet (highlighted in Fig. 1), is the independence of MNO profiles and provider changes. As depicted in Fig. 1, the provisioning server provides an IoT security service, which acts as a root Certificate Authority (CA).

III. Improvement Proposals for IoT SAFE

In this section we propose improvements to the design of IoT SAFE. These include changes to the provided services, and interface optimizations that lead to simpler applet implementations and better integration with the host software stack.

A. Cohesion of Hash and Signature Functionality

The single-responsibility principle [13] states that every class or service should have a single responsibility. The

design of IoT SAFE combines the functionality of hashing and signing into a composite service, which breaks the single-responsibility principle. The `Compute Signature` and `Verify Signature` commands offer a mode of operation (`full-text processing`) including hashing.

Further, merging the hash functionality into this mode, violates the principle of statelessness [14]. This principle advances the scalability by avoiding retention of session information. In order to allow hashing messages exceeding the interface's supported frame size, input chaining is required. Input and output chaining is used to convey long command and response data fields respectively [15].

1) Requirements from TLS: A TLS handshake engine requires hashing and signing functionality, however not in the combination provided by IoT SAFE. For the *CertificateVerify* message, signing the hash digest over all handshake messages sent and received until this point in time, is required. The `pad-and-sign processing` mode of IoT SAFE is sufficient for this use case. Theoretically, delegating the hashing to the applet is possible, but might not be done for the following three reasons.

First, hashing in the applet provides no additional security gain. Further, delegating this functionality leads to the necessity to transfer all received handshake messages to the applet. This results in a degraded performance, because hashing on the host controller is typically faster than transferring the messages. Utilizing a (D)TLS cipher suite with PSK authentication is a notable exception, since in this case the handshake is considerably shorter.

Second, during the *CertificateVerify* message generation, a signature over the hashed handshake messages is required. Therefore, the hash context needs to be cloned and finalized beforehand, as depicted in Fig. 3. Cloning the hash context is required, since subsequent handshake messages must be added to the same hash context in order to calculate the *verify_data* in the *Finish* messages. Overall, cloning the hash context for the handshake message digest is required twice during the TLS 1.2 handshake, highlighted in Fig. 3. The `Compute Signature` command interface does not allow cloning a corresponding hash context. This implies that the `full-text processing` mode cannot be used in this context of TLS.

Third, not all TLS libraries support the delegation of hashing. For instance *OpenSSL* allows the implementation of custom engines, which can be used to delegate ECC and RSA signatures to the applet. However, neither *OpenSSL* nor *mbedTLS* offer mechanisms to support the delegation of hashing.

2) Provide Separate Command for Hashing: Providing a dedicated command for hashing within the applet, results in the following benefits.

First, the commands to compute and verify signatures require only the `pad-and-sign` mode. Thus a single command is sufficient for signing and verifying pre-computed hashes. This implies that no input chaining is required for the `Compute Signature` command. Additionally, avoiding sessions allows merging the `Compute Signature – Init` and `Compute Signature – Update` commands, resulting in a simpler interface and less APDU transactions.

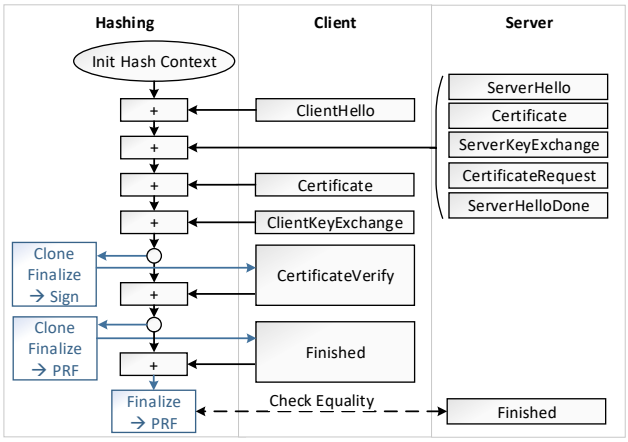

Fig. 3. Client-side Hashing Procedure during TLS 1.2 Handshake

TABLE I
APDU DATA FOR HASH COMMAND

APDU Field	Data type	Description
`CLA`	Byte	To be defined
`INS`	Byte	To be defined
`P1`	Byte	Session number
`P2`	Fragment flags	`80h` First `40h` Last
	Finalize flag	`20h` Finalize and return digest
Payload for first APDU	Optional TLV	Initial hash digest Value Number of bytes already hashed
	Mandatory TLV	Hash algorithm
	Plain data	Data to hash (can be empty)
Payload for following APDUs	Plain data	Data to hash (can be empty)

Second, support to clone a hash context is useful for resource-constrained TLS implementations without hashing capabilities. The session concept could be used to manage multiple hash contexts.

A hash command could be designed as shown in Table I. `P1` of the APDU specifies the session number, and `P2` is used to distinguish first, intermediate, and last APDUs in the session. For the first fragment, the payload consists of two optional Type-Length-Value (TLV) fields with the initial hash digest value and the number of bytes already hashed, to optionally support continuing a hashing operation. Afterwards, a mandatory TLV field, containing the hash algorithm to be used, and the first block of the data to hash follow. Intermediate and last fragment contain further message blocks to be hashed. With each APDU an intermediate hash can be exported, by setting the `Finalize and return digest` flag. Computing the intermediate hash does not discard the internal state of the hash session and further data might be added. The session is implicitly reset when an APDU is sent with `P2` indicating the first fragment.

B. Applet Store Improvements

The IoT SAFE applet store contains private keys, public keys, files, and secret keys. We propose having key pairs, consisting of a private and a public key, instead of storing them separately. This way, it is ensured that the metadata for both elements is identical. The `Object Access Conditions` would apply to the public key. Reading private keys should never be possible, and the `Cryptographic Functions` flag `Signature (generation or verification)` has to be split into two separate flags for generation and verification of signatures. Private keys should only be generated inside the applet, and never be imported via the management interface for security reasons. The generation of a private key without a public key is not possible in Java Card, only key pairs can be generated. That eliminates the need for having single private keys in the applet store.

Together with the introduction of key pairs, we propose changes to the interfaces of `Compute DH` and `Verify Signature`. For these operations, the public key data should be passed as payload instead of a reference to a public key object in the applet store. This eliminates the need for standalone public keys in the applet store and causes a reduction of required interface commands. Currently, a `Put Public Key` command precedes each `Compute DH` and `Verify Signature` command. The proposed change eliminates the need for two consecutive commands. We recognize the fact that public key data must still be stored in a Java Card `Public Key` object to be used for verify operations. However, it is possible to create public key objects that store data in transient memory. In addition, only one object for each supported public key type (e.g. RSA, ECC) is required to temporarily store public key data for verify operations.

The object metadata of keys contains a `Key Specific Usage` field. This field has currently no purpose in the applet, and could be removed. Information currently encoded in this field, which is used by the hosting device, could also be encoded as part of the `Identifier` or `Label`.

C. Command Simplification and Improvement

Several commands in IoT SAFE provide functionality to retrieve data or metadata from the applet. The related interfaces are not following the principles of single-responsibility and statelessness. We propose different interfaces that solve those problems, as well as defining more efficient encodings for data and metadata.

1) Merge `Read File` and `Read Public Key` Commands: From host controller perspective, public keys in IoT SAFE and files are equivalently. Both can only be read and can be larger than the maximum payload of a single APDU. Therefore, the separate `Read Public Key` command should be eliminated and the definition of the `Read File` command extended to allow reading public keys too.

Further, the structure definitions of public keys should be simplified such that there is only one layer of TLV, instead of currently up to three. This reduces the amount of code required for encoding and decoding the keys on the host, and subsequently the overall complexity.

TABLE II
APDU FOR GET OBJECT LIST COMMAND

APDU Field	Data type	Description
`CLA`	Byte	To be defined
`INS`	Byte	`CBh`
`P1`	Byte	`01h`
`P2`	Byte	Group index
Response Data	Optional TLV (in this order)	Private key identifiers Public key identifiers Secret key identifiers File identifiers
Status Word	Short	`6300h` End of list not reached `9000h` End of list reached

2) Reduce Scope of `Get Object List` Command: The `Get Object List` command provides a functionality to get the complete metadata from the applet, which is redundant to the other `Get Data` commands, and additionally it carries an internal state.

To remove these drawbacks, the command should return the `Identifier` of each object in a TLV structure. The metadata for this object is then retrieved via the respective `Get Data` command. Additionally, we propose to define a fixed ordering of these structures, and supply an index in `P2` as shown in Table II. To use the available space in an APDU as efficiently as possible, 11 (maximum size that fits in worst case) `Identifier` structures are grouped into one APDU. These groups are assigned with a group index, which is constant until the next update of the applet store.

To read the full object list, the host will query the applet by increasing the group index until it receives the `9000h` status code. By using a fixed group size of 11, the applet can efficiently compute the current position in the object list without the need to keep any state.

3) Parser Complexity Reduction: Results of commands to the IoT SAFE applet must be interpreted by the host controller. IoT SAFE uses TLV encoded data throughout this interface. In structures, such as the `Private key information structure`, some fields are conditionally present. This increases the complexity to parse these structures, because the conditions must be evaluated to verify the correctness of the received data structures. Additionally, in programming languages such as C, TLV encoded structures will be parsed to a matching `struct` type, which must contain all possible fields. To minimize the complexity of the parser on the host controller and the applet, we suggest to avoid conditional fields. They can either be encoded with a zero value for flags or in case of the `Label` field, with zero length.

The second problem increasing parser complexity, are inconsistencies in the definition of the command APDUs which use chaining. For the commands `Read Public Key` and `Get Object List` the bits controlling chaining are defined in `P2`, whereas for all other commands these bits are in `P1`. This definition necessitates two code paths for encoding and decoding APDUs instead of one generic solution. We propose

to define the chaining bits to always be in P2, which fixes this issue without any drawbacks.

4) Consistency and Versatility: To make the specification easier to understand and implement, the set of possible status words should be minimized. As an example, the 6A82h (*File not found*) status word is only used for the Read File command, while all other commands return 6985h (*Conditions of use not satisfied*) for objects that are not found. We propose to consistently use 6A82h (*File not found*) in case an object was not found in the applet store when referenced by its Identifier or Label.

During analysis of the specification, we discovered that it does not mention nor cover the case where the transport does not support the maximum APDU payload size for command and response. To resolve this, either the specification should be changed to support a flexible chaining protocol for all APDUs, or mandate that the transport must support the maximum payload size.

D. Supported Cryptographic Algorithms

The use cases for IoT SAFE are geared towards (D)TLS, but not all core algorithms to complete a (D)TLS handshake without a cryptography library on the host controller are defined. Therefore we propose the following adaptions to IoT SAFE.

1) HKDF Key Derivation for TLS 1.3: The specification defines the Compute HKDF command which implements the extraction step of the HMAC-based Key Derivation Function (HKDF) algorithm. This is not sufficient to perform a full TLS 1.3 key derivation without a cryptography library on the host, which means that the HKDF algorithm including a full Hash-based Message Authentication Code (HMAC) and hash algorithm implementation must be present on both, the host controller and the applet. We propose to extend the IoT SAFE specification with an interface for the HKDF-Expand step, which allows to remove the duplicated parts from the host controller.

2) PRF Key Derivation for TLS 1.2: An optional feature of the applet is the PRF SHA-256 key derivation for TLS 1.2. The Compute PRF command supports three modes of operation. In the PSK-Plain and PSK-ECDHE mode, a pre-master secret must be calculated before the Pseudo Random Function (PRF) calculation. For this calculation, the key data of the selected Secret Key object, which is stored in the applet store, must be available. Secret Key objects are typically implemented based on Java Card HMACKey objects that store the key material in protected memory. Extracting it into a less secured buffer for the pre-master secret calculation results in a security risk.

E. Default Applet Store Structure for TLS

In order to integrate the IoT SAFE functionality into a TLS library on the hosting device, the library needs to know how to select the appropriate objects in the applet store. Each object in the applet store has an Identifier and a Label used to select the object. For a typical TLS cipher suite such as TLS_ECDHE_ECDSA_AES_128_GCM_SHA256, the following objects are required in the applet:

- Key pair for client authentication
- Volatile key pair for key agreement
- Set of root certificates with at least one item

The listed objects are utilized during the handshake sequence as depicted in 2. The public key of the client authentication key pair is only required to be embedded in a Certificate Signing Request and in the client certificate, which is stored in a File object within the applet store. The private key is required during every TLS handshake to create the signature during the *CertificateVerify* message.

The volatile key pair for key agreement is required with the corresponding Cryptographic Functions flag. The Generate Key Pair function is used to generate a new key pair at every handshake. The public key is placed in the *ClientKeyExchange* message. The private key is used for the Compute DH function, together with the server's ephemeral public key from the *ServerKeyExchange* message to compute the pre-master secret.

With the current API without the proposed improvements of Verify Signature and Compute DH, temporary, volatile public key objects are required to store the public keys that are part of the server's certificate chain, and the server's public key for key agreement.

Finally, a set of root certificates, with at least one item, must be available in the applet store, to verify the last item in the server's certificate chain. These root certificates are stored in File objects.

We propose to define a default configuration of the applet store for that use case. The configuration should include object Identifier, Label, and other relevant metadata. This allows TLS implementations to rely on that configuration, which implies that an IoT SAFE integration into a TLS library can be reused by many applications.

IV. Evaluation

The evaluation is done regarding the following aspects: state-less and self-contained commands; scalability; interface efficiency; and missing features for TLS 1.3.

A. Comparison of Transmitted Number of APDUs

To evaluate whether the proposed improvements reduce the number of APDUs sent in a typical TLS handshake, we count the number of APDUs needed. We compare the current version of IoT SAFE to a version implementing the improvements from Sections III-A and III-C, depicted in Table III.

The selected TLS 1.3 cipher suite for this comparison is named TLS_AES_128_GCM_SHA256 with Elliptic Curve Digital Signature Algorithm (ECDSA) and Rivest-Shamir-Adleman (RSA) certificates. It is equivalent to TLS_ECDHE_ECDSA_AES_128_GCM_SHA256 and TLS_ECDHE_RSA_AES_128_GCM_SHA256 in TLS 1.2. To minimize the number of variables the following assumptions were taken: hashing of messages is done on the host controller, an ECDSA certificate fits into two APDUs (512 bytes)

TABLE III
IoT SAFE APDU Count for TLS 1.2 Handshake Messages

TLS Message	IoT SAFE Command	ECC		RSA	
		Old	New	Old	New
ClientHello	Get Random	1	1	1	1
(Server) Certificate	Read File	2	2	4	4
	Put Public Key	2	0	3	0
	Verify Signature	2	1	3	4
ServerKeyExchange	Put Public Key	2	0	3	0
	Verify Signature	2	1	3	4
(Client) Certificate	Read File	2	2	4	4
ClientKeyExchange	Generate Key Pair	1	1	1	1
	Put Public Key	2	0	3	0
	Compute DH	1	1	1	1
	Compute PRF	1	1	1	1
CertificateVerify	Compute Signature	2	1	3	2
(Client) Finished	Compute PRF	1	1	1	1
(Server) Finished	Compute PRF	1	1	1	1
Total APDU count		**22**	**13**	**31**	**24**

and an RSA certificate fits into four APDUs (1024 bytes), the public key of the root CA is not cached in a key slot. Caching the public key of the root CA would remove two (ECC) or three (RSA) APDUs from the total amount, but with the drawback of more complicated provisioning and updating.

The comparison shows that our improvements are able to significantly reduce the number of APDUs sent for the case where ECC certificates are used. For the case with RSA certificates it is also reduced, but not as significantly, because the larger public keys and signatures for RSA require more data to be transferred.

B. Use Case Analysis of PSK Mode

The `PSK-Plain` and `PSK-ECDHE` modes are intended for TLS 1.2 with a PSK-based cipher suite. This scenario is unlikely for resourceful IoT devices with a 4G modem, and software stacks to control that modem. Even if a PSK-based cipher suite is utilized, an implementation of the TLS handshake engine and record layer is still required on the hosting device. These modes would be useful for resource-constrained systems, if the applet provides an implementation of the TLS handshake engine, as certain proprietary HSMs, such as the Infineon OPTIGA™ Trust X, do.

C. Feasibility Analysis of Hash Context Cloning

Implementing the cloning of a hash context is currently not possible with the Java Card `MessageDigest` API, which means a Java Card Application Programming Interface (API) extensions is required, or hashing must be implemented within the applet. The proprietary HSMs Infineon OPTIGA™ Trust X and Microchip ATECC608B provide this hash context cloning functionality.

V. Conclusion and Future Work

In this work, we evaluated the IoT SAFE applet and suggested improvement proposals in order to reduce complexity, increase flexibility, and ease the integration into TLS libraries. The evaluation shows, that certain proprietary HSM implementations have advantageous design aspects compared to the IoT SAFE specification. Further, the performance regarding required APDU count during the TLS handshake, can be increased tremendously.

Future work aims to influence future specification and designs decision for standardized interfaces in order to support easier and more efficient integration of hardware security. In the course of the funded project *ADACORSA*, we will implement the applet including improvements into a drone platform for demonstration purposes.

VI. Acknowledgment

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876019. The JU receives support from the European Unions Horizon 2020 research and innovation programme and Germany, Netherlands, Austria, Romania, France, Sweden, Cyprus, Greece, Lithuania, Portugal, Italy, Finland, Turkey.

References

[1] I. Analytics, "Cellular IoT and LPWA Tracker (Q4 2020)," Tech. Rep., 2020, [Online; accessed 2021-03-22]. Available: https://iot-analytics.com/product/cellular-iot-lpwa-connectivity-market-tracker-2020-25-update-q4-20/

[2] Q. Liao, T. Fischer, J. Gao, F. Hafeez, C. Oechsner, and J. Knode, "A Secure End-to-End Cloud Computing Solution for Emergency Management with UAVs," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.

[3] Marcus Janke, Dr. Peter Laakmann, in *Attacks on Embedded Devices*. Embedded World Conference Nurenberg, 2016.

[4] I. Corporation, "Intel Secure Device Onboard," Tech. Rep., 2017, [Online; accessed 2021-04-12]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/idz/iot/briefs/sdo-product-brief.pdf

[5] GSMA, "IoT SAFE Executive Summary," Tech. Rep., 2020, [Online; accessed 2021-02-16]. Available: https://www.gsma.com/iot/wp-content/uploads/2020/05/IoT-SAFE-Executive-Summary.pdf

[6] ——, "SGP.22 RSP Technical Specification," Tech. Rep., 2020, [Online; accessed 2021-03-22]. Available: https://www.gsma.com/esim/wp-content/uploads/2020/06/SGP.22-v2.2.2.pdf

[7] ——, "IoT Security Applet Interface Description," Tech. Rep., 2019, [Online; accessed 2021-04-09]. Available: https://www.gsma.com/iot/wp-content/uploads/2019/12/IoT.05-v1-IoT-Security-Applet-Interface-Description.pdf

[8] Microchip, "CryptoAuthLib - Microchip CryptoAuthentication Library," Tech. Rep., 2021, [Online; accessed 2021-03-31]. Available: https://github.com/MicrochipTech/cryptoauthlib

[9] NXP, "SE050 APDU Specification," Tech. Rep., 2021, [Online; accessed 2021-04-12]. Available: https://www.nxp.com/docs/en/application-note/AN12413.pdf

[10] I. T. AG, "OPTIGA Trust X Software Framework," Tech. Rep., 2020, [Online; accessed 2021-03-31]. Available: https://github.com/Infineon/optiga-trust-x

[11] T. Fischer, D. Pirker, C. Lesjak, and C. Steger, "TinyI2C - A Protocol Stack for connecting Hardware Security Modules to IoT Devices," in *2020 International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom)*, 2020, pp. 1–7.

[12] D. Pirker, T. Fischer, C. Lesjak, and C. Steger, "Global and Secured UAV Authentication System based on Hardware-Security," in *2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 2020, pp. 84–89.

[13] E. Chebanyuk and K. Markov, "An approach to class diagrams verification according to SOLID design principles," in *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2016, pp. 435–441.

[14] R. T. Fielding and R. N. Taylor, "Architectural Styles and the Design of Network-Based Software Architectures," Tech. Rep., 2000.

[15] I. GlobalPlatform, "Card Specification ISO Framework," Tech. Rep., 2014, [Online; accessed 2021-04-12]. Available: https://globalplatform.org/wp-content/uploads/2014/03/GPC_ISO_Framework_v1.0.pdf