Revisiting Multi-Factor Authentication Token Cybersecurity: A TLS Identity Module Use Case

Pascal Urien
LTCI, Telecom Paris
19 Place Marguerite Perey 91120 Palaiseau, France
Pascal.Urien@Telecom-Paris.fr

***Abstract*—Multi-factor authentication (MFA) procedures are widely used by digital systems. There are usually performed by hardware tokens comprising a microcontroller and an USB interface. The security level is increased by computing cryptographic procedures in secure elements such as smartcards. Authenticity of MFA token is a critical topic since hardware or software components may be cloned or modified, for example through supply chain. Due to industrial competition cyber security aspects of MFA token are not generally in the public domain, and therefore somewhat relies on security by obscurity (SbO). In this paper we present an original MFA token built with open hardware (Arduino) and javacard, which realizes a TLS pre-shared-key identity module (TLS-IM). The microcontroller is authenticated by SRAM dynamic PUF features, its software is checked by attestation procedure based on the bijective MAC time stamped algorithm. The javacard application is authenticated by PKI means, and manages a TLS-PSK channel for remote administration.**

***Keywords*—** Security,Secure Element, IoSE, TLS

I. Introduction

Multi-Factor Authentication (MFA) is a technique [1][2] that enables the computing of cryptographic procedures involved in authentication processes, thanks to authentication credentials bound to human user, according to several factors. For example, something user has, something user known, something user does.

As an illustration FIDO (*Fast IDentity Online*) standards define authentication protocols, supporting a second-factor hardware authenticator. The reference [3] lists some of such hardware tokens, which typically comprise an USB interface, a microcontroller and a secure element. This paper also reviews side channel attack [4] performs on the *Google Titan Security Key* that computes ECDSA signature based on the "*comb*" algorithm used for scalar multiplication.

A secure element [6][7] is a tamper resistant microcontroller, widely used in bank card, SIM module or electronic passport. It typically performs *Key Management System* (KMS), in a trusted computing environment. Nevertheless secure elements, according to common criteria (CC) standards, have different *evaluation assurance level* (EAL), and may be approved by various organizations (for example EMVco).

From a cyber security point of view, the authenticity of MTA token is an important topic. The device can be cloned by malicious manufacturers, or modified during supply chain journey. Genuine component can be replaced by counterfeit secure element or microcontroller [5]. Software integrity is obviously a critical requirement; backdoors enabling fraudulent interactions with secure element or malicious use of KMS, can be implemented in microcontroller or secure element.

The contribution of this paper is the secure design of a MFA token (TLS identity module TLS-IM [8] [9] [10]), used for authentication purposes with TLS1.3 pre-shared-key servers. The TLS-IM token is based on open technologies; it comprises an Arduino microcontroller and a javacard. This use case avoids *security by obscurity* (SbO) and enables clear description of cyber security features whose main goals are:

- to provide three-factor authentication (3FA, something you have, something you know, something you do)

- to authenticate the microcontroller using PUF (Physical Unclonable Function)

- to check the integrity of the microcontroller software

- to authenticate the secure element and its content

This paper is organized according to the following outline. Section 2 introduces TLS-IM identity modules, security requirements, and practical use. Section 3 details TLS-IM hardware components (Arduino and javacard) and microcontroller software. Section 4 describes secure element application downloading, configuration and authentication. Section 5 presents microcontroller enrollment and authentication according to dynamic SRAM PUF procedure. Section 6 describes microcontroller software attestation procedure working with the bijective MAC time stamped algorithm. Finally Section 7 concludes this paper.

II. TLS Identity Module (TLS-IM)

The last version (TLS 1.3) of the well known TLS protocol provides two authentication methods for server and client: signatures such as ECDSA (*Elliptic Curve Digital Signature*) based on asymmetric keys, and shared symmetric secret (named pre-shared-key, PSK). As depicted in [8] [9] associated procedures may be computed in dedicated identity module (i.e. TLS-IM). In this paper we focus on cryptographic functions required by TLS-PSK, on the client side.

A. TLS-PSK Cryptographics procedures

We are using the following notations; HL16: hash Length, 16 bits; HL8: hash length, 8 bits; H0: hash(empty); hash: SHA256.

Four symmetric keys (ESK, DSK, BSK, FEK) are computed from PSK (32 bytes value), according to the following relations:

ESK= HMAC(salt=0,PSK)

DSK=HMAC(ESK,HL16||0d746c7331332064657269766564||HL8||H0||01)

BSK=HMAC(ESK,HL16||10746c73313320657874206269e646572||HL8||H0||01)

FEK=HMAC(BSK,HL16||0E746C7331332066696E69736865640001)

Two procedures that we name *binder* and *derive* are required for TLS authentication method based on PSK:

- binder (data)= HMAC(FEK, data), in which *data* a hash (SHA256) value, computes an authentication value in the TLS first flight

- derive(DHE)= HMAC(DSK, DHE) computes the handshake (HS) secret from the Diffie Hellman exchange (DHE) secret.

B. Security requirements

The main idea of TLS identity module (TLS-IM) is to support *binder* and *derive* procedures (defined in previous section) in tamper resistant computing environment, such as secure element (SE). Two-factor authentication (2FA), for example a PIN code, is a classical protection against malicious use of lost or stolen TLS-IM token. Secure element communication interfaces are specified by ISO7816 standards [6], while smart phones or laptops usually support Bluetooth RFCOMM or Serial USB communication port. Therefore the TLS-IM token includes a microcontroller named *Secure Element Processor* (SEP, [10]) that realizes a logical bridge with ISO7816 protocols. The SEP or SE can be modified or replaced by supply chain attacks. Therefore we would like to check the integrity of SEP software, and establish that SEP and SE are genuine devices. Because the terminal to which is plugged the TLS-IM token can include malicious software, we introduce a third factor authentication (3FA), a push button that enables the *derive* function during a time slot (5 seconds) notified by a blinking LED.

C. Integration in TLS stack

TLS-IM modules require two callback functions for invoking *binder* and *derive* procedures. We added these procedures in the *wolfSSL* open library; they work either over PC/SC (*Personal Computer/ Smart Card*) API dedicated to smartcard readers (2FA case, i.e. TLS-IM module is a smartcard), or serial port (3FA case, i.e. TLS-IM token with SEP and SE).

III. TLS-IM Token Components

A. Hardware

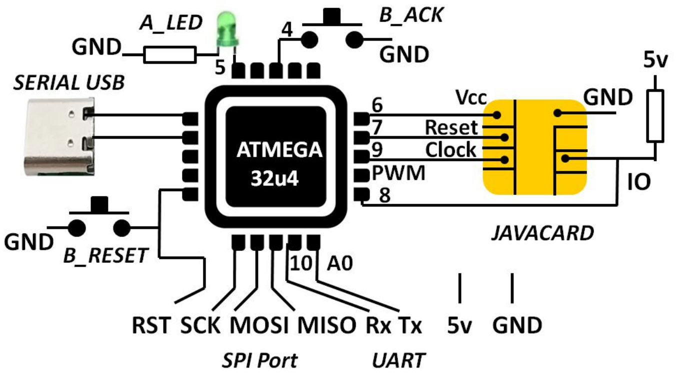

Fig. 1. The TLS-IM Token simplified architecture

The TLS-IM token (see figure 1) comprises two components: a javacard (J3R180, Javacard 3.0.5, 85KB FLASH, 3KB SRAM) and a microcontroller (ATMEGA 32u4) based on RISC technology clocked at 16MHz, with 32 KB FLASH, 2.5 KB SRAM, 1 KB EEPROM, and USB 2.0 full-speed/low-speed interface.

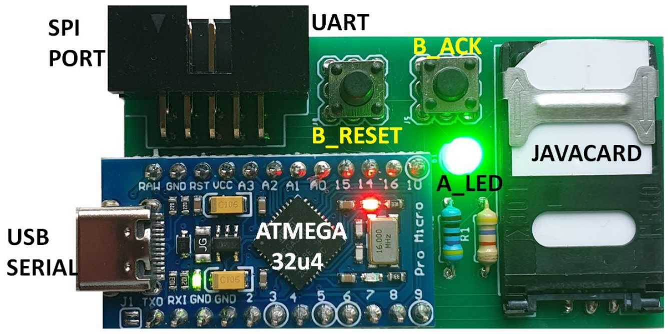

Fig. 2. The experimental TLS-IM token

The javacard is connected to the micro controller thanks to the ISO7816 five wires interface (Vcc, Gnd, Reset, Clock, IO). Two communication interfaces are available: serial USB and UART. Serial USB is dedicated to laptops or smart phones supporting OTG (*USB On-The-Go*) technology. UART (*Universal Asynchronous Receiver Transmitter*) is used for SRAM PUF (*Physically Unclonable Function*). We observed that serial USB uses quite all available SRAM when the device is plugged to an USB port, what modified SRAM content after power up.

The user interface (UI) comprises a LED (acknowledgment LED, A_LED) and two push buttons (reset button B_RESET, and acknowledgment button B_ACK). Blinking A_LED indicates that an action is required on B_ACK. Simultaneous use of B_RESET and B_ACK reboots the device in administrator mode, for which all implemented commands are available.

Figure 2 illustrates the experimental device including an Arduino pro micro 32u4 board, a SIM socket (2FF format), two buttons, a LED, and a 10 pins SPI (*Serial Programming Interface*) connector used for flashing the micro controller (thanks to signals RST, SCK, MOSI, MISO), powering up the device, and communicating through the UART.

B. *Software*

The token software is designed with the Arduino integrated development environment (IDE), a dedicated ISO7816 library [11] drives the secure element. It manages two serial communication channels (channel 1: Serial USB and channel 2: UART), it has two working modes user and administrator, and supports acknowledged commands (thanks to the B_ACK button).

cmd	comment	ch	ack	adm
dump	Dump 1KB SRAM	1-2	no	no
bmac SEED	Compute bijective MAC time stamped	1	no	no
on	Power on SE	1	no	no
off	Power off SE	1	no	no
user PIN	Power on SE & Send PIN	1	no	no
binder DATA	Compute binder	1	no	no
derive DATA	Compute derive	1	yes	no
getpk	Read SE public key	1	no	no
getcert	Read SE certificate	1	no	no
auth DATA	Authenticate SE	1	no	no
apdu DATA	Send APDU to SE	1	no	yes
ewrite ADR DATA	Write data in EEPROM	1	no	yes

Fig. 3. Main commands of the TLS-IM token

The software provides a SHELL (see figure 3) that parses ASCII command sent over serial interfaces. Channel 2 is used to dump SRAM content for PUF operations; channel 1 processes all other commands. The user's plane realizes the bMAC algorithm, the two TLS-IM procedures "*binder*" and "*derive*" (protected by the B_ACK button), and starts the smartcard with a PIN. Secure element authentication is performed thanks to three commands: "*getpk*", "*getcert*", and "*auth*".

The administrator mode is entered by the simultaneous press of B_RESET and B_ACK BUTTON. It enables the "*apdu*" command, which is required to exchange ISO7816 APDU packets with the secure element, needed for TLS-PSK setting or javacard application downloading. It also gives access to the "*ewrite*" command that controls writing operation in EEPROM.

IV. Secure Element

The secure element stores the pre-shared-key (PSK), and computes "*binder*" and "*derive*" procedures. The observed computing time for "*binder*" and "*derive*" procedure is about 100ms. A wrong content (bad PSK) induces a denial of service (DoS) risk. From the administration point of view the two issues to be taken into account are: 1) software downloading, and authentication, and 2) secure PSK storage.

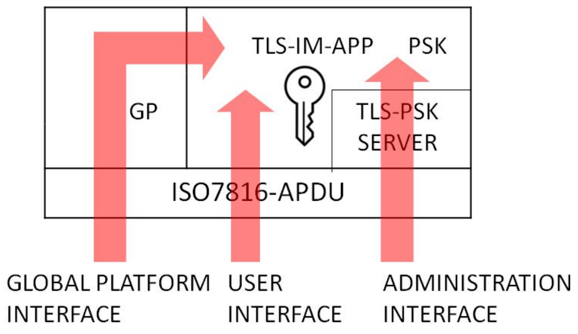

Fig. 4. TLS-IM-APP application in javacard environnment

A. *TLS-IM APP Downloading*

Most of secure elements implement Global Platform (GP) protocols, which perform secure software downloading, protected by two symmetric (128 bits) secrets, from which are derived two keys used for encryption and integrity purposes. Because GP transport is based on APDU (see figure 4), the TLS-IM token provides, in administration mode, a transparent bridge for its support.

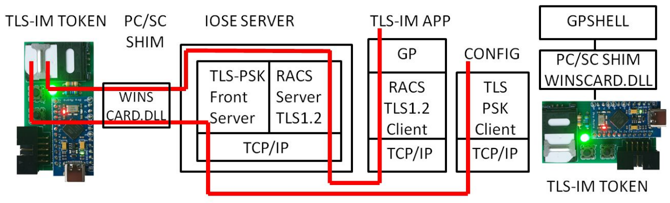

Fig. 5. Software downloading in TLS-IM token SE, local mode with GPSHELL (right part), remote mode with IOSE server (left part)

Open software such as GPSHELL [12] based on PC/SC API can be used with a SHIM (implemented for example in a WINSCARD.DLL for Windows) that translates PC/SC calls over channel 1 serial port (see figure 5, right part). Such SHIM also enables to plug TLS-IM token to IOSE server [10] (see figure 5 left part), as explained in next section.

B. *TLS-IM APP Configuration*

Upon instantiation the TLS-IM-APP software creates a pair of public and private key over the SECP256k1 elliptic curve. According to [13] the application manages an internal TLS-PSK server (see figure 4) initialized with a provider pre-shared-key (Provider-PSK) used for administration. This server is associated to a name, the *Secure Element Name* (SEN [10]).

TLS-IM tokens are supported, thanks to PC/SC SHIMs, by IOSE server [10], which maintains two networks interfaces over TLS. First is used by RACS (*Remote APDU Call Secure*) daemon, typically for GP support. Second provides a front TLS-PSK interface with a backend named TLS server (such as TLS-IM token).

The TLS-PSK internal channel is used for two tasks:

-reading the secure element public key and forwarding a certificate (the ECDSA signature of the public key hash, by a Certification Authority);

- setting the TLS-IM pre-shared-key.

C. *TLS-IM-APP authentication*

In the user mode the TLS-IM token provides three commands for SE authentication:

- *getpk* reads the secure element public key

- *getcert*, reads the secure element certificate

- *auth* DATA, computes an ECDSA signature of DATA value with the secure element private key

V. Secure Element Processor (SEP) Authentication

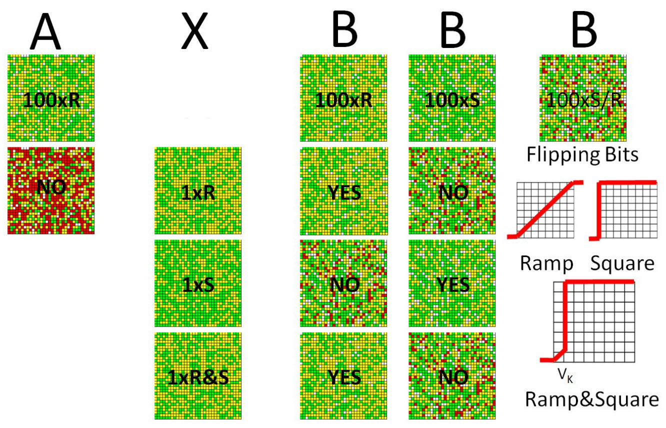

Fig. 6. Authentfication of ATGEMA32u4 micocontrollers. SRAM contents,references, based on 100 measures, have been recorded for 2 devices A et B. A device is authenticated with different powerup signals such as Ramp (R), Square (S), and Ramp&Square (R&S). Memory cell colors are the following, yellow always 0, green always 1, white,noise, red fipping bits..

The idea behind SEP authentication is to detect counterfeit devices. We want to identify and to authenticate microcontroller used by TLS-IM token.

A. *SRAM PUF and SRAM dPUF*

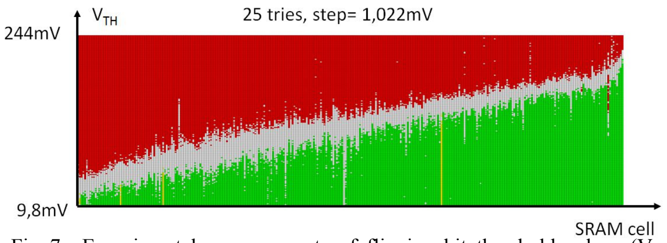

Fig. 7. Experimental measurements of flipping bit threshold values (V_{TH}), thanks to Ramp&Square signals with different knee voltage (V_K, from 9,8mV to 244mV, with step of 1,022mV). Yellow always 0 (25 tries), green always (25 tries) 1, red flipping bits (25 tries), noise between 1-24 for 25 tries. SRAM cells are ordered by increasing V_{TH}.

Microcontroller chips embed SRAM memory, a set of cells made with six CMOS transistors realizing two logical inverters head to tail. Upon power up, most of cells take a fix value (due to physical dissymmetry), while other have random content. This effect is called SRAM-PUF. Furthermore [14][15], due to capacitance dissymmetry, the supply voltage rise time may create flipping bits. Such memory cells take a fix value (either 0 ou 1), but with inverted values depending on the supply voltage rise time. Flipping bits are not observed for low rising time (less than 10V/s), and are created for rising time of about 100V/s. This effect occurs at a given voltage threshold (V_{TH}). The observed V_{TH} values (see figure 7) are in the range of a few hundred mV; they are measured thanks to R&S power-up signals, with different knee (V_K) values (Vk is defined in figure 6).

The token is power-up by three kinds of signals: Square, (S, about 1500V/s), Ramp (R, 5V/s) and Ramp and Square (R&S, the voltage knee V_K is at 625mV). Only square signal creates flipping bits. Dynamic PUF (dPUF) relies on the fact that S or R&S signal cannot be differentiated by the microcontroller. Even if PUF bits are known, a random use of S and R&S powering up, creates a set of SRAM-PUF that cannot be guessed by a malicious firmware.

B. *SEP Enrollment & Authentication*

The token is powered by a dedicated generator built with an ATMEGA2560 (Arduino, 16 MHz) comprising a digital to analog (DAC) converter (MCP4725, with 12 bits resolution) and an operational amplifier (LM358P), which can sink about 40mA. The firmware that controls output voltage processes 4 samples per ms; the rising time ranges between 1500V/s and 5V/s. When the powering up process (P) is complete, one KB of SRAM is dumped thanks to a serial interface.

The powering process is repeated n times (P_n). Three types of cell rams are identified (see figure 6): always zero (Z, colored in yellow), always one (O, colored in green), noise (N, colored in white) sometimes one or zero. So the memory M is divided in three sets: M= O U Z U N.

We call domain (D) the set D= O U Z.

In order to compare two memory contents M_1 and M_2, we define:

-The common domain, $C_D = D_1$ n D_2 ($C_D = M_{1,2}$ U $F_{1,2}$)

-The matching bits in C_D, $M_{1,2}$ = (O_1 n O_2) U (Z_1 n Z_2)

-The flipping bits in C_D, $F_{1,2}$ = (O_1 n Z_2) U (Z_1 n O_2)

-The similarity factor S_f= #$M_{1,2}$/#C_D (# being the cardinal of a given set). S_f is used as metric for device authentication, with a value around 0,99.

Matching bits (see figure 6) are colored either in yellow or green. Flipping bits are colored in red. Other bits (noisy bits) are colored in white.

For a given microcontroller, flipping bits belong to a common domain created by S and R powering up signals; depending on the powering up signal (S, R, R&S), they are located either in O or Z.

A device (k) identification compares $P_{1,k}$ (single power on) with different $P_{n,i}$ (n power on) for devices i. About 50% of flipping bits are expected for wrong device, while quite no flipping bits are observed (i.e. about 100% of matching bits are expected, so S_f> 0,99) for the right device.

For two different devices (i and k) comparison between $P_{n,i}$ and $P_{n,k}$ creates about 50% (S_f=0,494) flipping bits in the common domain (see figure 8)

For the same device, comparison of Pn obtained with S powering signal to P'n collected with R or R&S signals enable to identify flipping bits (see figure 9)

Device	Domain	Zero	One	Flip ping
#10	3873	2162	1711	228
#11	3867	1963	1904	182
Common Domain	3658	1045	807	1852

Fig. 8. Common Domain for two devices,using 100 measures profile, the memory size is 4096 bits (512 bytes)

Powering Signal	Common Domain	Zero	One	Flip ping	S_f	Ma tch
S - R100	3873	1723	1699	451	0,884	no
S - S100	3591	1506	2084	1	0,999	yes
RS- R100	3873	2162	1708	3	0,999	yes
RS- S100	3591	1503	1748	340	0,905	no

Fig. 9. Device #10 authentication with S and R&S signals, the memory size is 4096 bits (512 bytes)

VI. Software Attestation

Software integrity is a major concern, increased by supply chain journey. Malicious software enables multiple MIM (man in the middle) attacks. The bijective MAC time stamped (bMAC_TS) is a software-based [18] remote attestation [17] [18], which comprises a prover (i.e. bMAC_TS) and a (human) verifier. Verifier sends challenge to prover, which computes and returns a response.

A. Bijective MAC TimeStampted (bMAC_TS)

Given a memory of size m (including FLASH, EEPROM and SRAM), bMAC computes a memory fingerprint (h, such as SHA256 or Keccak256) according to a P permutation.

bMAC = h(A(P(0)) || A(P(1) || … || A(P(m-1))

A(x) is the byte content associated to an address x. The computing time (CT) is returned with the bMAC

In Z/pZ* (multiplicative group of integers modulo p), with p (p>m) a safe prime, p=2q+1with q St Germain prime, and p=7 modulo 8, we define P as :

$$P(y) = F(y-1) - 1 , y \in [0, p-2]$$

$$F(x) = g_2^{s_1 g_1^{x} \bmod p} \bmod p,\ x, s_1 \in [1, p-1]$$

g_k are generators in Z/pZ* defined according to

$$g_k = p - (2^k \bmod p) , k \in [1, q-1]$$

The number of P permutations is $(p-1)(q-1)^2$, about $m^3/4$ (2^{43} for a 32 KB memory). Therefore bMAC responses (whose total size is about 2^{43}x32bytes) cannot be stored in the token memory. Nevertheless malicious software may perform memory copy attacks, which use copy of genuine memory. In order to detect such event, the computing time is measured by an internal timer using the microcontroller clock (16MHz) sub frequency (16MHz/64); stopping and restarting this timer creates random time measurement error (in the order of 4 µS per *stop&start* action).

B. Enrollment & Authentication

The TLS-IM token software works with the prime p=36887 (32K + 2,5K + 1K = 36352). Given a positive integer of 31 bits inserted in the "*bmac*" command, a pseudo random generator [19] computes s_1, g_1 and g_2 values. The hash function is Keccak256; the whole content of the FLASH and EEPROM memories, and part of SRAM memory used by Keccak256 context are included in MAC calculations. Because the EEPROM memory is not used, a dedicated command ("*ewrite*") enables to modify its content; the idea being to avoid memory copy attack. In the same spirit, the unused portion of FLASH memory (storing code), is filled with pseudo random bytes (inserted in the software image file).

Multiple calls of "*bmac*" commands with different seed enable to build a table, indexed by seed values, which stores bMAC and CT. As illustrated by figure 8, the computing time repartition looks similar to a normal law. For 722 tries, in unit of 4µs (the internal timer resolution) the minimum is 6.762.590, the maximum 6.783.464, the average 6.774.474 (27 seconds), and the standard deviation 3005 (12 ms). If we admit that quite all computing times are observed in three standards deviation, the resulting entropy is about 13 bits.

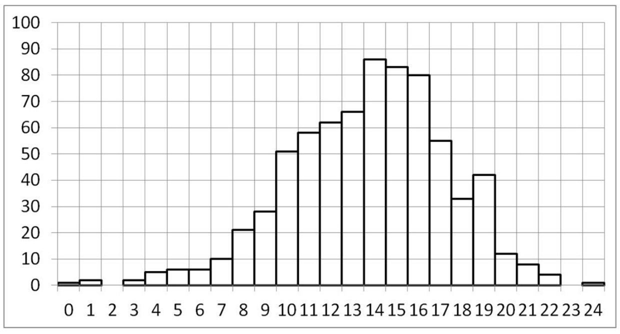

Fig. 10. bMAC computing time distribution (722 measures) for TLS-IM token. Computing times are divided in 25 classes between minimum to maximum value.

VII. Conclusion

In this paper we introduced 3FA TLS-IM token made with a microcontroller, a secure element, a LED and two push buttons. Codes were written with Arduino IDE [20] and Oracle Javacard 3.0.5 Software Development Kit.

The secure element should be manufactured by a trusted company. It is protected by GP keys required for application downloading, asymmetric keys managed by application provider for software authentication, and Provider-PSK initialized by application provider in order to handle TLS-PSK secure channel for administration purposes.

The microcontroller is authenticated by PUF technique (dPUF) and its software integrity is checked by an attestation procedure (bMAC_TS).

The 3FA is realized by the token, a PIN code, and a push button for cryptographic procedure authorization.

We believe that this design, based on open hardware and software technologies could be applied in many use cases for which trusted multi-factor authentication is required.

References

[1] A. A. S. AlQahtani, Z. El-Awadi and M. Min, "A Survey on User Authentication Factors," 2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 2021, pp. 0323-0328, doi: 10.1109/IEMCON53756.2021.9623159.

[2] Ometov, A., Bezzateev, S., Mäkitalo, N., Andreev, S., Mikkonen, T., & Koucheryavy, Y. (2018). "Multi-factor authentication: A survey". *Cryptography*, *2*(1), 1.

[3] Lomme, V., "An Overview of the Security of Some Hardware FIDO(2)" TokensHardwear.io Security Trainings and Conference, Netherlands, 202224-28 October 2022, https://hardwear.io/netherlands-2022/presentation/security-of-Hardware-FIDO(2)-tokens.pd

[4] Thomas Roche, Victor Lomné, Camille Mutschler, Laurent Imbert. "A Side Journey To Titan: Revealing and Breaking NXP's P5x ECDSA Implementation on the Way." USENIX Security 2021, USENIX Security Symposium, Aug 2021, Virtual, Canada. pp.231-248.

[5] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor and Y. Makris, "Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain," in Proceedings of the IEEE, vol. 102, no. 8, pp. 1207-1228, Aug. 2014, doi: 10.1109/JPROC.2014.2332291.

[6] Jurgensen T.M., Guthery, S.B., "Smart Cards: The Developer's Toolkit", O'Reilly

[7] Chen, Z., "Java Card™ Technology for Smart Cards, Architecture and Programmer's Guide", ADDISON-WESLEY, 2000

[8] IETF Draft, "Identity Module for TLS Version 1.3", draft-urien-tls-im-09.txt, July 2023

[9] P. Urien, "Innovative TLS 1.3 Identity Module for Trusted IoT Device," 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2021, pp. 1-4, doi: 10.1109/CCNC49032.2021.9369656.

[10] P. Urien, "On Line Secure Elements: Deploying High Security Keystores and Personal HSMs," 2023 International Conference on Computing, Networking and Communications (ICNC), Honolulu, HI, USA, 2023, pp. 450-455, doi: 10.1109/ICNC57223.2023.10074066.

[11] ISO7816 Library For Arduino, https://github.com/purien/SCLIB-ARDUINO

[12] Global Platform Shell, https://github.com/kaoh/globalplatform

[13] IETF Draft "Internet of Secure Elements", draft-urien-coinrg-iose-07.txt, IETF Draft, April 2023

[14] Elshafiey, A. T., Zarkesh-Ha, P. and Trujillo, J. (2017). *The effect of power supply ramp time on SRAM PUFs*. 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS). https://doi.org/10.1109/MWSCAS.2017.8053081.

[15] Urien, P. (2020). Innovative Dynamic SRAM PUF Authentication for Trusted Internet of Things. 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). https://doi.org/10.1109/WiMob50308.2020.9253432.

[16] Sigurd Frej Joel Jørgensen Ankergård, Edlira Dushku, Nicola Dragoni, "State-of-the-Art Software-Based Remote Attestation: Opportunities and Open Issues for Internet of Things", Sensors 2021, 21(5), 1598; https://doi.org/10.3390/s21051598

[17] Seshadri, A., et al (2006). SCUBA: Secure Code Update By Attestation in sensor networks. WiSe '06: Proceedings of the 5th ACM workshop on Wireless security. https://doi.org/10.1145/1161289.1161306

[18] Urien, P. (2020). Proving IoT Devices Firmware Integrity With Bijective MAC Time Stamped. IEEE 6th World Forum on Internet of Things (WF-IoT). https://doi.org/10.1109/WF-IoT48130.2020.9221395.

[19] Park, S.K., Miller, K.W, (1998) Random Numbers Generators: Good Ones Are Hard to Find. Communication of ACM , Volume 31, Number 10, pp1192-1201.

[20] https://github.com/purien, seen November 2023