An Innovative Security Architecture for Low Cost Low Power IoT Devices Based on Secure Elements

A four quarters security architecture

Pascal Urien
Telecom ParisTech, LTCI, University of Saclay
23 avenue d'Italie, 75013, Paris, France

***Abstract*—This demonstration presents a low cost, low power object based on Arduino board, whose security is enforced by a secure element. Most of IoT devices are built over a General Purpose Unit (GPU) associated to a radio System on Chip (SoC), and controlling multiple sensors and/or actuators. We designed a similar architecture, in which communications are secured by a TLS server running in a secure element. This secure chip performs strong mutual authentication with remote clients dealing with X509 certificates, and acts as an identity module.**

Keywords— Internet of Things; Security; Secure Element;

I. INTRODUCTION

According to [1], the Internet of Things is a network of connected things. Security is undoubtedly a prerequisite for social acceptance of such infrastructures, and for *Cyber Physical Systems* (CPS) safety requirements. As an illustration a recent paper [3] demonstrated an attack on smart bulbs, whose manufacturer, according to its 2016 annual report, is the market leader of connected home lighting. This attack performs remote reprogramming from drones or cars, and enables the injection of virus. These bulbs are built over ATmega2564RFR2 SoCs, comprising 8-bit microcontroller, 256KB of FLASH, 32KB of SRAM, AES hardware accelerator, and IEEE 802.15.4 radio transceiver. It relies on two threats: first a bug on Zigbee stack which is reset to factory and afterwards joins a malicious network; second the embedded AES key is recovered thanks to a *Correlation Power Analysis* (CPA) attack. Because these devices share the same key, malicious software updates (encrypted by the AES-CCM algorithm [2]) can be performed.

In this demonstration we present a secure low cost object, motorized by an Arduino board (cf. figure 2) and interacting with a mobile application. The security relies on a secure element, with a smartcard form factor. It is enforced by an unique TCP server using the well known 443 (TLS) port. The TLS stack runs in the secure element (SE) and performs a strong mutual authentication based on X509 certificates between end entities. So the SE works like a tamper resistant firewall that authenticates incoming connections. More generally the demonstrated object realizes a four quarters secure architecture, built over four components (cf. figure 1). A *General Purpose Unit* (GPU) coordinates three devices, 1) a SoC in charge of communications, 2) a secure element performing TLS protocol operations and defining object identity [4], 3) sensors and actuators controlled by the GPU.

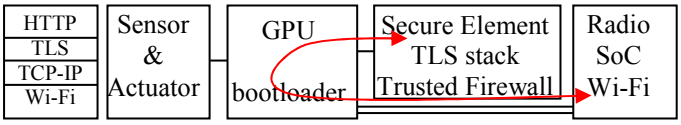

Fig. 1. A four quaters security architecture, based on secure element.

II. DEMONSTRATION HARDWARE

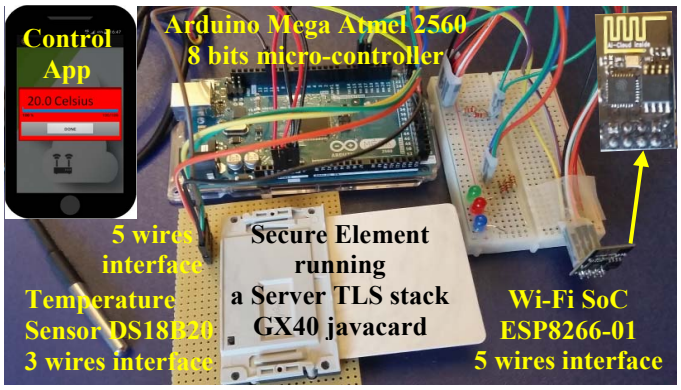

Fig. 2. The demonstration hardware and the mobile control application.

A. The GPU: Arduino Mega.

The GPU is an Arduino Mega board, based on ATmega2560 8-bit micro-controller, clocked at 16 MHz, and comprising 256 KB of FLASH, 4 KB of EEPROM, and 8 KB of SRAM. The processor has no operating system, and is initialized from a 8KB bootloader; it controls numerous digital input/output pins, and provides PWM (*Pulse Width Modulation*) facilities. Communications with Wi-Fi SoC, secure element, and temperature sensor are based on serial links. The limited SRAM size, is the most critical resource.

B. Identity Module and TLS stack : Secure Element

Secure elements are defined by ISO7816 standards and are usually supporting *Java Virtual Machine*, running software written in the *javacard* language [5][6]. From a hardware

978-1-5386-4790-5/18/$31.00 ©2018 IEEE

point of view an ISO7816 device has 5 pins: Vcc, ground, clock, IO and reset. The chip is directly powered by an Arduino digital output, and is activated on demand. The clock is generated thanks to a PWM facility at the 4 MHz frequency, leading, according to the ISO7816, to a basic bit time of 372/4 MHz (about 93µs). Nerveless this time (named ETU) is divided by two in the working mode (372 /2/ 4MHz) thanks to an ISO7816 procedure called PTS (*Protocol Type Selection*). A byte needs 12 bits (1 start, 8 data, 1 parity, 2 stops), so the useful throughput is about 1792 bytes/s. The ISO7816 driver (cf. figure 3) manages the single serial IO pin and uses a timer resource (a 16 bit counter with a resolution of 1/16 µs) to sample incoming bit stream and to generate outgoing binary signal. It implements the T=0 protocol widely used by SIM cards. A request comprises a five bytes header; the secure element echoes the second byte (INS), and afterwards outgoing bytes are forwarded to SE or incoming bytes are received from SE; in both cases the SE ends its response by a two byte status (SW1 SW2). The secure element has 70KB FLASH and 4KB SRAM available for applications. The TLS stack written for *javacard*, requires a memory fingerprint of about 30KB. It is able to process a TLS full session opening in 2 seconds, plus an extra delay needed for the transfer of about 2500 bytes, i.e. 1,4s for a serial throughput of 1800 bytes/s.

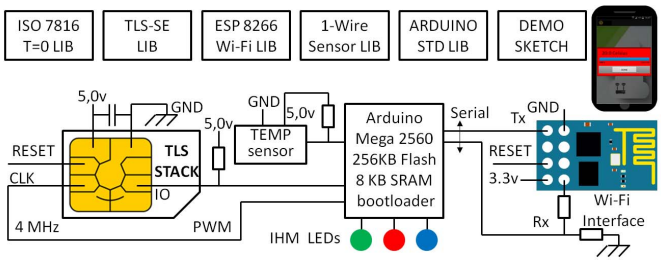

Fig. 3. Software Architecture, and remote control from a mobile.

C. Communication Module: Wi-Fi SoC

The Wi-Fi SoC (ESP 8266) comprises two parts, an analog area dealing with radio resources, and a processor with at the most 36KB of SRAM associated to an external SPI FLASH, up to 4 MB. The chip implements the IEEE 802.11i security protocol and provides a TCP/IP stack with client and server features. It is managed thanks to well known AT commands, from a serial interface of which we fix the baud rate to 38400 bauds. On the Arduino side the reception and transmission buffer sizes are respectively set to 256 and 128 bytes. The reception (Rx) link is monitored by a dedicated (and unique) procedure, all incoming events such as TCP session opening and closing, data reception, data transmission confirmation, and errors, are notified by dedicated messages. The size of the reception serial buffer (256 bytes) allows a maximum delay of about 65ms before a fatal buffer overflow. A SRAM buffer of 2500 bytes is available (on the Arduino side) for the storage of incoming and outgoing TLS packets. The TLS handshake is organized in flights (4 flights for a full session [4][6]) whose maximum size is about 1500 bytes in our platform. This memory area enables the transfer of TLS messages between network and secure element.

D. Sensors & Actuators

Our device is equipped with DS18B20 digital temperature sensor, manufactured by the Dallas Semiconductor company. The sensor works over a proprietary 1-Wire® protocol, which enables the use of multiple devices.

III. Sofware Architecture

As illustrated by figure 3 the software architecture includes 5 libraries and the demonstration sketch (e.g. the main C module in the Arduino terminology). We developed the original TLS-SE LIB that drives TLS packet exchanges between the SoC and the secure element. The ISO 7816 LIB and ESP Wi-Fi LIB are deeply modified versions of existing libraries, mainly to take into account a share SRAM buffer and critical timing issues. One-Wire sensor LIB and Arduino LIB are standard components. The demonstration sketch is working according to the following logical organization. The SoC serial receiving line is periodically pooled in order to detect a TCP connection event; the scheduler collects data from the temperature sensor and manages a LED (green) blinking. When an incoming connection is detected the first TLS flight (*Client Hello*, less than one hundred byte) is received. The serial Rx buffer size (256 bytes) is sufficient to guaranty the storage of this packet. Thereafter the secure element (SE) is power on, and initialized. The 1st TLS flight is pushed to the SE thanks to the ISO7816 LIB. The SE returns over the ISO7816 IO the 2nd TLS flight (*Server Hello*) whose size is about 1500 bytes. This flight is afterwards sent over the TCP session. The SoC receiving line is pooled until the completion of the reception of the 3rd TLS flight (*Client Finished*) whose size is about 1000 bytes. This block is thereafter pushes to the SE over its IO pin, thanks to the ISO7816 LIB services. The SE returns the 4th flight (*Server Finished*), which is thereafter sent over the TCP session. The TLS session is opened and transferred [6] from SE to the Arduino microcontroller. An HTTPS request is received from the SoC serial line; a parser returns an HTTPS response (including the collected temperature), which is sent over the SoC serial line. Upon completion the TCP session is closed. In order to display information about the system state the blue led blinks (with a 50 ms delay) each time a flight is sent over TCP, when a packet is received over TCP, or before forwarding ISO7816 message to the secure element.

References

[1] K. Pretz,"The Next Evolution of the Internet", http://theinstitute.ieee.org, 2013.

[2] C. O'Flynn, "A Lightbulb Worm? Details of the Philips Hue Smart Lighting Design", White Paper, Black Hat USA 2016.

[3] E. Ronen, C. O'Flynn, A. Shamir, A. Weingarten, "IoT Goes Nuclear: Creating a ZigBee Chain Reaction", 2017 IEEE Symposium on Security and Privacy

[4] IETF draft, "TLS and DTLS Security Modules", draft-urien-uta-tls-dtls-security-module-04.txt, June 2017

[5] P. Urien, "Securing The IoT With TLS/DTLS Server Stacks Embedded In Secure Elements: An ePlug Usecase", IEEE CCNC 2017.

[6] P. Urien, "Introducing TLS/DTLS Secure Access Modules for IoT frameworks: Concepts and experiments", IEEE ISCC 2017