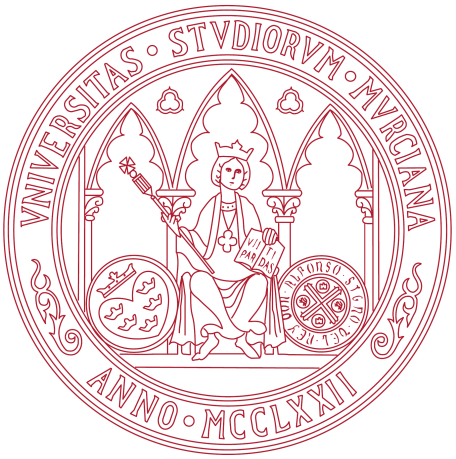
INTEGRACIÓN DE IDEMIX EN ENTORNOS DE IOT

JOSÉ LUIS CÁNOVAS SÁNCHEZ

Tutores
ANTONIO FERNANDO SKARMETA GÓMEZ
JORGE BERNAL BERNABÉ



Facultad de Ingeniería Informática
Universidad de Murcia

José Luis Cánovas Sánchez: *Integración de Idemix en entornos de IoT*

Junio 2017

Ohana means family.
Family means nobody gets left behind, or forgotten.

— Lilo & Stitch

Dedicated to the loving memory of Rudolf Miede.

1939 – 2005

ABSTRACT

Short summary of the contents in English... a great guide by Kent Beck how to write good abstracts can be found here:

https://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [7]

ACKNOWLEDGMENTS

Put your acknowledgments here.

Many thanks to everybody who already sent me a postcard!

Regarding the typography and other help, many thanks go to Marco Kuhlmann, Philipp Lehman, Lothar Schlesier, Jim Young, Lorenzo Pantieri and Enrico Gregorio[1], Jörg Sommer, Joachim Köstler, Daniel Gottschlag, Denis Aydin, Paride Legovini, Steffen Prochnow, Nicolas Repp, Hinrich Harms, Roland Winkler, Jörg Weber, Henri Menke, Claus Lahiri, Clemens Niederberger, Stefano Bragaglia, Jörn Hees, and the whole LaTeX-community for support, ideas and some great software.

Regarding LyX: The LyX port was intially done by *Nicholas Mariette* in March 2009 and continued by *Ivo Pletikosić* in 2011. Thank you very much for your work and for the contributions to the original style.

1 Members of GuIT (Gruppo Italiano Utilizzatori di TeX e LaTeX)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

IoT Internet of Things

ZKP Zero-Knowledge Proof

P2ABCE Privacy-Preserving Attribute-Based Credentials Engine

PoC Proof of Concept

1 INTRODUCTION

In recent years some new concepts have appeared in common people's vocabulary, like *machine learning*, *big data*, *artificial intelligence*, *automation*, etc., but there are two in particular that we are going to focus and try to combine: Internet of Things (IoT) and Internet Security & Privacy.

The IoT is a term with a wide range of interpretations [3], but a breaf definition could be the set of devices, mainly resource constrained, that are interconnected between them in order to achieve a goal. This includes from lampposts with proximity sensors that talk to each other in order to light up part of the street when a passerby walks by, to a sensor on your clothes that tells the washing machine how much detergent to use.

Security & Privacy, thanks to organizations like WikiLeaks, are now taken in consideration by any technology consumer, not only professionals. People are conscious about what their data can be used for, demanding more control over it.

And IoT has proved to not address neither security nor privacy, with recent events like the Mirai botnet DDoS attack on October 2016, considered the biggest DDoS in history [10], or like the multiple vulnerabilities affecting baby monitors [14].

A recent approach to address the problem of privacy is the *strong anonymity*, that conceals our personal details while letting us continue to operate online as a clearly defined individual [5]. One very promising way to achieve this is using Zero-Knowledge Proofs (ZKPs) , cryptographic methods that allows to proof knowledge of data without disclosing it. Furthermore, IBM has been developing a cryptographic protocol suite for privacy-preserving authentication and transfer of certified attributes based on ZKP, called Identity Mixer, Idemix for short [6].

To read more about ZKP aside the introduction done in this thesis, you can read my Mathematics thesis [15].

The goal of this project is to integrate Idemix with the IoT. It will be done using the ABC4Trust's Privacy-Preserving Attribute-Based Credentials Engine (P2ABCE), a framework that defines common architecture, policy language and data artifacts, but based on either IBM's Idemix or Microsoft's U-Prove [13]. This gives us a standardized language to exchange Idemix's messages between IoT devices and usual PCs.

2

STATE OF THE ART

In this chapter we present the two dimensions of this project: the IoT development state and an introduction to IBM's privacy-preserving solution, Idemix.

2.1 INTERNET OF THINGS

The development for Internet of Things depends heavily on each target device. We can differentiate two big groups: those with enough processing power to act like an usual computer, and those constrained devices that can't perform arbitrary tasks, sometimes called *embedded*.

We can consider in the first category powerful ARM devices like Raspberry Pi 3, with a 64-bit architecture, 4 CPU cores, 1GB of RAM, which can even compile its own binaries, run the *Java Virtual Machine*, etc., working in practice like any other computer. These kind of devices do not present any major difficulty in terms of research.

What we will consider to be a more *pure* IoT device will be the constrained ones, where it's not trivial to develop any algorithm and run it successfully.

Very known devices fall into this category, like Arduino, powered by Atmel's AVR ATmega328 8-bit microprocessors, with 32KB of program flash memory, 2KB of SRAM, 16MHz of CPU [2]. It seems clear that memory and computation power are a very big issue to deal with when developing to this devices.

A step above in power we can find ESP8266, the most famous Espressif's microcontroller, with built-in WiFi antenna, a Tensilica 16bit RISC microcontroller at 80MHz, 50Kb of RAM, and 1MB flash memory [4]. The possibility of direct WiFi connectivity is its best selling point, putting the *Internet* in Internet of Things.

In another level of power we have microcontrollers usually found in routers, but used in many other applications, like the On-Board Units (OBU) used in Vehicular ad hoc networks (VANET). Characteristics in this range vary around a single core 32-bit CPU, at some hundreds MHz, with tens or hundreds MB of RAM and flash memory, which places them near the first IoT mentioned category.

Although one can code in assembly language for these microcontrollers, there exist C compilers, and many frameworks to build firmware binaries: Arduino Core, Contiki, propietary SDKs, Mongoose OS, ThreadX OS (Real-Time OS), OpenWrt, LEDE, etc. Each firmware targets specific ranges of devices, depending on processing power and memory limitations. For example, Arduino and Contiki aim for microcon-

trollers like Atmel's ATmega and TI's MSP430, but can also be used in ESP8266, a more powerful microcontroller.

In particular OpenWrt and LEDE (a fork of OpenWrt) are based on Linux, with optimized library binaries, providing many packages through *opkg* [11]. To compile C/C++ code, build the firmware or packages, a complete build system and cross-compiler toolchain can be installed in a x86 host, and using Makefiles select the target hardware [12].

Devices running OpenWRT and other Linux distros are in the limit between IoT categories, but the need of a cross-compiler marks that they belong to the second category.

Starting a big project development for IoT aiming the most constrained devices may not be a good idea. The lack of usual operative system tools (like POSIX), I/O, or even threads can make debugging a tedious task. With good programming practices one can start from the top and slowly end at the bottom with reliable code.

For this reason the current Proof of Concept (PoC) is developed on LEDE, Linux Embedded Development Environment [8], using the Onion Omega2 development board, a Mediatek MT688 microcontroller [9] with a 32-bit MIPS 24KEc CPU at 580MHz, 64MB of RAM and 16MB of flash and built-in WiFi. This development board uses LEDE as its firmware, but its CPU is also listed as compatible with ThreadX OS [16], a Real-Time Operative System for embedded devices.

The PoC will take advantage of the Linux system using files and sockets like in any other Linux desktop distribution, so we can focus on the project itself rather than the specific platform APIs for storage and connectivity.

2.2 IDEMIX

TODO: MOVE TO MOTIVATION The problem of Internet privacy has been approached by securing the transmission channel (e.g. SSL/TLS) and the data stored in both ends (strict access policies, local encryption, etc.). In the end, the data exists in two entities, the owner of the data and the service provider. The owner is the most interested in securing his data, and can apply as many measures as he wants, but only on his side of the table. The service provider that stores the user data needs it to provide the service, and a successful attack would reveal many users data, aside from how many measures each one used to protect it. The case of PlayStation Network outage in 2011 [1] affected 77 million accounts, with suspected credit card fraud, is an example of this kind of attacks.

Other solutions are based on minimal disclosure. Standards like OAuth offer secure delegated access to the user information and

when registering to a new service, the user can give a key to access only the data they want from another trusted service. This lets the service provider to work with the OAuth server, offering the same service as before without knowing as much data.

But this is only minimizes how many services have our data. Our OAuth provider could be attacked, revealing all our data, or our service provider, revealing now less data.

IBM proposes a step forward using Zero-Knowledge Proofs:

> *If your personal data is never collected, it cannot be stolen.*

TODO: PONER DESCRIPCIÓN + FORMAL DE IDEMIX, LUEGO P2ABCE, Y METER QUE EN P2ABCE HABÍA UNA IMPLEMENTACIÓN DE SC BASADA EN C

3 OBJECTIVES AND METHODOLOGY

3.1 PROJECT DESCRIPTION

3.2 WORKING METHODOLOGY

3.3 DEVELOPMENT ENVIRONMENT

3.3.1 *Hardware*

To test our design in a realistic but easy to work with deployment, we used as the IoT device an Onion Omega2 development board, and as the delegation server a Raspberry Pi 3.

Onion Omega2

ONION OMEGA2: A device that falls inside the second category of IoT, powerful enough to run a familiar Linux environment, where we can develop and debug the first PoCs without troubling ourselves with non-related to the project problems.

Nonetheless, the Omega2 needs fine tunning to start operating, and basic knowledge of electronics is vital to make it work. The two main things to begin with Omega2 are:

- A reliable 3.3V with a maximum of 800mA power supply (a USB with a step-down circuit works fine), with quality soldering and wires to avoid unwanted resistances.
- A Serial to USB adapter wired to the TX and RX UART pins to use the Serial Terminal, in case WiFi doesn't work and no SSH is available, and because the connection is more reliable in case of wireless interferences.

MCU	Mediatek MT688 [9]
CPU	MIPS32 24KEc 580MHz
RAM	64MB
Storage	16MB
Firmware	LEDE (OpenWRT fork distro)
Connectivity	Wifi b/g/n
Power	3.3V 300mA

Table 1: Onion Omega2 Specifications.

Raspberry Pi 3

RASPBERRY PI 3: Another familiar environment, powerful enough to debug and hold the delegated P2ABCE Java services (User, Verifier, ...) of P2ABC with its 1GB of RAM, and with two network interfaces it's perfect to work as the gateway for the IoT devices to the Internet.

Only a microSD with enough space to burn the binary with the OS is needed to plug&play with the Raspberry Pi. We use Raspbian, a stable Debian based distro, recommended by the Raspberry Pi designers, and ready to use with the P2ABCE compiled *.jar* services.

CPU	ARMv8 64bit quad-core 1.2GHz
RAM	1GB
Storage	microSD
Firmware	Raspbian (Debian based distro)
Connectivity	Wifi n + Ethernet
Power	5V 2A

Table 2: Raspberry Pi 3 Specifications.

3.3.2 *Software*

The development is divided between the IoT device code and the P2ABCE services.

The P2ABCE is already written in Java, and few modifications will be done to the code in comparison to the existing project size, so we will continue using Java with the P2ABCE part.

All IoT devices, have a C cross-compiler, some even a C++ cross-compiler. The worst case scenario is that one must write assembly code, and that code will be specific of that target, so we won't consider them. If now we focus on the most constrained devices, we could find out that some can't compile C++, some may not have many common libraries, and that the memory limitations they face make practically impossible to use dynamic memory, if we want to avoid very possible execution malfunctions.

For that reason, the developed code for IoT devices must be written with standard C without using dynamic memory.

A project with thousands of lines of code can't be written in a single file. And to manage the compilation of multiple files, organized in various directories, we will use CMake.

CMake has many advantages over Makefiles:

- Cross-platform. It works in many systems, and more specifically, in Linux it generates Makefiles.
- Simpler syntax. Adding a library, files to compile, set definitions, etc. can be done with one CMake command, with rich documentation on the project's website.
- Cross-compilation. With only a CMAKE TOOLCHAIN file, CMake sets up automatically the cross-compilation with Makefiles and the C/C++ cross-compiler provided.

Although the ideal final code is pure C without external libraries or dynamic memory, the PoC uses three major libraries:

- OpenSSL: Provides reliable and tested AES and SHA256 implementations.
- LibGMP: Provides multiprecision integer modular arithmetic.
- cJSON: Provides a JSON parser to store and read the status in a human readable way.

These three libraries are used to implement different interfaces in the project, and C implementations of these interfaces should replace the external libraries in the future.

Finally, we use Docker to deploy the compilation environments:

P2ABCE ENVIRONMENT A container with OpenJDK 7 and Maven installed, with the Idemix maven plugins installed following the project instructions to use Idemix as the Engine for P2ABCE.

LEDE SDK ENVIRONMENT A container with CMake and the LEDE SDK [8] installed and configured for the Omega2 target.

The Dockerfiles can be found in the Appendix.

4

DRAFT

4.1 APDU

4.2 P2ABCE EXISTING CODE

In the P2ABCE repository [13] is available the project's code, divided in two solutions: a complete P2ABCE implementation in Java and a Multos Smartcard implementation as companion for the project.

The Java code is managed by a Maven project, structured using various known design patterns, but not of our interest. The structure we are actually interested in are the REST Services and their use of the Components classes, in which the smartcards are included.

P2ABCE project is based on the concept of smartcards to store the credentials, logical or physical. An interface is defined to communicate with these smartcards, and then different implementations allow to use either *Software Smartcards* or *Hardware Smartcards*.

The *SoftwareSmartcard* class implements the interface in Java, suitable for tests and self-stored smartcards that any application using P2ABCE may need.

The *HardwareSmartcard* class uses the standard APDU messages [TODO:ref] to interact with smartcards. P2ABCE defines for every method in the mentioned interface, the necessary APDU instructions, and currently relies on *javax.smartcardio* abstract classes (implemented by Oracle in their JRE) to communicate with the smartcard reader. This way, it doesn't matter what manufacturer issues the smartcard, or if it's an Android device, if they support the APDU API, P2ABCE will work with them.

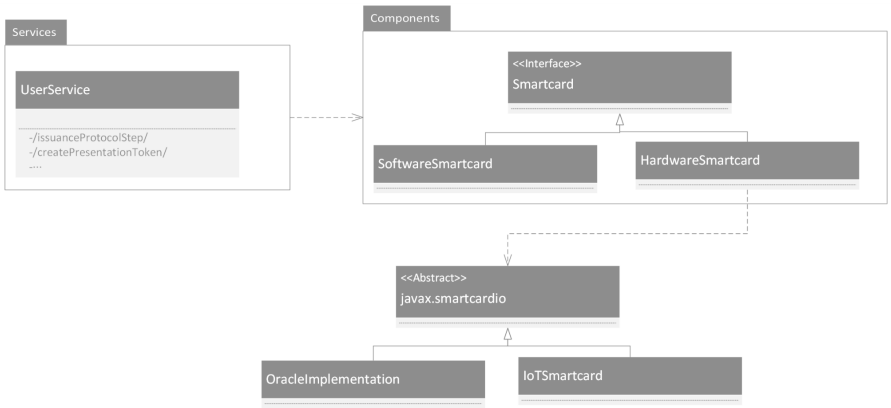


Figure 1: Basic P2ABCE structure

As a PoC the P2ABCE project includes the ABC4Trust Card Lite, an implementation for ML3-36K-R1 Multos Smartcards. The code is written in C, but is very dependent on the Multos framework, aside from numerous bugs and bad coding habits. The code is structured in two files, *main.h* and *main.c*, with 557 and 5157 lines of code respectively. The *main()* function is a 2635 lines long *switch-case* with practically no comments.

APPENDIX

APPENDIX TEST

Lorem ipsum at nusquam appellantur his, ut eos erant homero concludaturque. Albucius appellantur deterruisset id eam, vivendum partiendo dissentiet ei ius. Vis melius facilisis ea, sea id convenire referrentur, takimata adolescens ex duo. Ei harum argumentum per. Eam vidit exerci appetere ad, ut vel zzril intellegam interpretaris.

More dummy text.

A.1 APPENDIX SECTION TEST

Test: Table 3 (This reference should have a lowercase, small caps A if the option `floatperchapter` is activated, just as in the table itself → however, this does not work at the moment.)

LABITUR BONORUM PRI NO	QUE VISTA	HUMAN
fastidii ea ius	germano	demonstratea
suscipit instructior	titulo	personas
quaestio philosophia	facto	demonstrated

Table 3: Autem usu id.

A.2 ANOTHER APPENDIX SECTION TEST

Equidem detraxit cu nam, vix eu delenit periculis. Eos ut vero constituto, no vidit propriae complectitur sea. Diceret nonummy in has, no qui eligendi recteque consetetur. Mel eu dictas suscipiantur, et sed placerat oporteat. At ipsum electram mei, ad aeque atomorum mea. There is also a useless Pascal listing below: Listing 1.

Listing 1: A floating example (`listings` manual)

```
for i:=maxint downto 0 do
begin
{ do nothing }
end;
```

BIBLIOGRAPHY

[1] *2011 PlayStation Network outage*. https://en.wikipedia.org/wiki/2011_PlayStation_Network_outage.

[2] *ATmega328 Datasheet*. http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf.

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey." In: *Computer Networks* 54.15 (2010), pp. 2787 –2805. ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.comnet.2010.05.010. URL: http://www.sciencedirect.com/science/article/pii/S1389128610001568.

[4] *Espressif ESP8266 Datasheet*. https://espressif.com/en/products/hardware/esp8266ex/resources.

[5] Tom Henriksson. "How 'strong anonymity' will finally fix the privacy problem." In: *VentureBeat* (2016). https://venturebeat.com/2016/10/08/how-strong-anonymity-will-finally-fix-the-privacy-problem/.

[6] *Identity Mixer*. https://www.research.ibm.com/labs/zurich/idemix/.

[7] Donald E. Knuth. "Computer Programming as an Art." In: *Communications of the ACM* 17.12 (1974), pp. 667–673.

[8] *Linux Embedded Development Environment*. https://lede-project.org/.

[9] *MediaTek MT7688 Datasheet*. https://goo.gl/kqD2gF.

[10] R. Thandeeswaran N. Jeyanthi. *Security Breaches and Threat Prevention in the Internet of Things*. IGI Global, 2017.

[11] *OPKG Package Manager*. https://wiki.openwrt.org/doc/techref/opkg.

[12] *OpenWrt's build system*. https://wiki.openwrt.org/about/toolchain.

[13] *P2ABCEngine*. https://github.com/p2abcengine/p2abcengine.

[14] Mark Stanislav and Tod Beardsley. *HACKING IoT: A Case Study on Baby Monitor Exposures and Vulnerabilities*. Tech. rep. Rapid7, 2015.

[15] José Luis Cánovas Sánchez. "Pruebas de Conocimiento Cero y sus Aplicaciones."

[16] *THREADX OS Real-Time OS*. http://rtos.com/products/threadx/.

DECLARATION

Put your declaration here.

, Junio 2017

José Luis Cánovas Sánchez

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of May 1, 2017 (`classicthesis`).