Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Integration of Anonymous Credential Systems in IoT constrained environments

JOSE LUIS CANOVAS SANCHEZ, JORGE BERNAL BERNABE, AND ANTONIO F. SKARMETA

Department of Information and Communications Engineering
Computer Science Faculty, University of Murcia, Spain
{joseluiscanovas, jorgebernal, skarmeta}@um.es

This work has received funding from IBM 2015 Faculty Award for Cyber Security and the European Union's Horizon 2020 research and innovation programme under grant agreement No 700085 (ARIES project).

ABSTRACT The pervasive nature of Internet of Things entails additional threats that compromise the security and privacy of IoT devices and, eventually, the users. This issue is aggravated in constrained IoT devices equipped with minimal hardware resources. Current security and privacy implementations need to be re-designed and implemented maintaining its LoA, aiming for this family of devices. To cope with this issue, this paper proposes the first novel attempt to leverage Anonymous Credential Systems (ACS) to preserve the privacy of autonomous IoT constrained devices. Concretely, we have designed a solution to integrate IBM's Identity Mixer into constrained IoT ecosystems, endowing the IoT with ACS's privacy-preserving capabilities. The solution has been designed, implemented and evaluated proving its feasibility.

INDEX TERMS Anonymous credential system, Identity Mixer, Internet of Things, Zero-knowledge proof

I. INTRODUCTION

Internet of Things is a term with a wide range of interpretations [3], briefly, we can think of it as billions of devices, mainly resource constrained, which are interconnected between them and the Internet, in order to achieve a goal.

The *anytime, anything and anywhere* nature of the IoT raises serious security and privacy concerns, since highly sensitive information will be exchanged and managed in common IoT scenarios. Nowadays, more and more people are starting to become aware of the privacy issues they need to face in these scenarios, which entails the use of a great amount of personal data. This includes not only the data shared with others, where one must trust they will keep it safe, but also the data collected about us and which we do not have direct control over it. Thus, one of the biggest challenges in IoT lies in the design of secure and privacy-preserving services, which will be deployed in everyday scenarios as an essential factor for the development of Smart Cities.

IoT is characterized by its interaction based on M2M (Machine to Machine). In traditional M2M environments, the issues about security and privacy have already been treated deeply, but in the IoT ecosystem, due to it's recent and fast growth, and its constrained nature, it still lacks of the mechanisms to address the privacy problems. To cope with this aim, a recent approach is the concept of *strong anonymity*, that conceals user's personal details while letting them to continue operating online as a clearly defined individuals [16]. Thus, a way to perform authentication and authorization in the most privacy-friendly approach need to be addressed. In this sense, Anonymous Credentials Systems (ACS) [9] and selective disclosure techniques allow to control what information can be revealed to others.

The ABCs (Attribute-Based Credentials) are used in the ACS as mechanism to cryptographically link the digital signature by an Issuer on a list of attribute-value pairs [19]. The most straightforward way for a user to convince a Verifier of her list of attributes would be to simply transmit her credential to the Verifier. With anonymous credentials, the user never transmits the credential itself, but rather uses it to convince the Verifier that her attributes satisfy certain properties - without leaking any information about the credential other than the chosen properties to display.

Indeed, apart from showing the exact value of an attribute, ACS enables that a user can even convince the Verifier that some complex predicate over the attributes holds, e.g. that

her birth date was more than 18 years ago, without revealing the real date. With classical symmetric and asymmetric cryptography it seems rather impossible to create such credentials without an explosion of signatures over every possible combination of attributes. For this reason, current solutions rely on Zero-Knowledge Proofs (ZKP), cryptographic methods that allow to proof knowledge of some data without disclosing it.

Based on ZKPs, IBM developed Identity Mixer[1], Idemix for short, a protocol suite for privacy-preserving authentication and transfer of certified attributes. It allows user authentication without divulging any personal data. Users have a personal certificate with multiple attributes, but they can choose how many to disclose, or only give a proof based on their values. Thus, no personal data is collected that needs to be protected, managed, and treated by third parties.

So far, Idemix or privacy-ABCs have been successfully applied to deal with traditional Internet scenarios, in which users can authenticate and prove their attributes against a service provider. However, due to the reduced computational capabilities of certain IoT devices, it has not been yet considered for IoT scenarios. As it is presented in the state of the art section, current implementations of Idemix are based on Java, which requires high computational and memory resources to be executed. To the best of our knowledge, this is the first proposal that tries to apply an solution for privacy-preserving authentication and authorization to IoT constrained scenarios, based on Anonymous Credential Systems.

The design and implementation presented in this paper will allow constrained IoT devices to carry out all actions available in the ACS systems and concretely in Idemix protocol, being in control of the decisions to take in every step. This achieves oblivious interactions between any traditional machine using Idemix and the IoT devices, without having to adapt the protocol, or a subset it, to interact with these new entities.

This document is structured as follows: Section II provides a state of the art analysis through the history of Idemix and related works, analysing what is of the most interest for the IoT perspective. Section IV presents a formal design of the proposed IoT and Idemix solution. Section V describes the PoC implementation developed. After any implementation, it is a must to validate it, as showed during the performance tests in section VI. Finally, our conclusions and lines for future work are described in section VII.

II. STATE OF THE ART

In this section we present a showcase of competing ACS solutions, focusing their application for the IoT, where the two most notable alternatives to Idemix are Persiano's ABC systems and Microsoft's U-Prove.

In 2004, Persiano and Visconti presented a non-transferable anonymous credential system that is multi-show and for which it is possible to prove properties (encoded by a linear Boolean formula) of the credentials [20]. Unfortunately, their proof system is not efficient since the step in which a user proves possession of credentials (that needs a number of modular exponentiations that is linear in the number of credentials) must be repeated k times (where k is the security parameter) in order to obtain a satisfying soundness.

Based on Persiano's proofs, an anonymous authentication for privacy-preserving IoT was presented in [2], but the studied performance analysis was carried out using Java on a traditional desktop PC, theoretically assuming IoT devices to be proportionally 40 times slower than the testing machine.

In 2000, Stefan Brands provided the first integral description of the U-Prove technology in his thesis [8], after which he founded the company Credentica in 2002 to implement and sell this technology. Microsoft acquired Credentica in 2008 and published the U-Prove protocol specification [7] in 2010 under the Open Specification Promise4 together with open source reference software development kits (SDKs) in C# and Java. The U-Prove technology is centered around a so-called U-Prove token. This token serves as a pseudonym for the prover. It contains a number of attributes which can be selectively disclosed to a verifier. Hence, the prover decides which attributes to show and which to withhold. Finally there is the token's public-key, which aggregates all information in the token, and a signature from the issuer over this public-key to ensure the authenticity [21].

Independently of the mentioned technologies, Jan Camenisch, Markus Stadler and Anna Lysyanskaya studied in [10], [11], [13] the cryptographic bases for signature schemes and anonymous credentials that later became IBM's Identity Mixer protocol specification [1].

Luuk Danes in 2007 studied theoretically how Idemix's User role could be implemented using smart cards [14], identifying what data and operations should be kept inside the device to perform different levels of security. The User role was divided between the smart card, holding secret keys, and the Idemix terminal, that commanded operations inside the smart card, or read the keys in it to perform the instructions itself. The studied sets were a combination of possibilities where the smart card would give all the information to the terminal, only partial information, or keep everything secret and perform all the private operations within itself.

Later, in 2008 Víctor Sucasas also studied an anonymous credential system with smart card support [17], equivalent to a basic version of Idemix, using a simulator to test the PoC and pointing out some crucial implementation details for performance. The researching tendency starts to show that smart cards are the best solution to hold safely the User's credentials.

In 2009, some Java smart card PoC for Idemix were developed in [6] and [22], but they weren't optimal and didn't include some Idemix's functionalities, like selective disclosure.

[1] Identity Mixer - https://www.zurich.ibm.com/identity_mixer/

In 2013, Vullers and Alpar, implemented an efficient smart card for Idemix [23], aiming to integrate it in the IRMA[2] project, and comparing the performance with U-Prove's smart cards. This new implementation was written in C, under the MULTOS platform for smart cards, and describes many decisions made during the development to improve the performance on such constrained devices. The terminal application was written in Java and used an extension of the Idemix cryptographic library to take care of the smart card specifics.

Later, the P2ABCE[3] project extended the concept of smart cards, physical or logical, as holders of the credentials. The P2ABCE project is a language framework that unifies different cryptographic ABC engines and policies, currently supporting U-Prove and Idemix. The Idemix library was updated to support P2ABCE and their last version is therefore interoperable with U-Prove. The smart card specification from the P2ABCE project can be considered the official version to work with. The Identity Mixer team instructs the use of P2ABCE in order to use Idemix itself.

Related to the IoT, the P2ABCE project has been used to test in a VANET[4] scenario how an OBU (On Board Unit) with constrained hardware could act as a User in a P2ABCE ecosystem [15]. However, after the theoretical analysis, the paper only simulates a computer with similar performance as an OBU, without adapting the existing Java implementation of P2ABCE to a real VANET system.

Another recent approach to integrate Idemix in the IoT was performed in [5], where it can be seen as a first attempt of a real re-implementation, not simulation, but aimed for Android devices, written in Java, and therefore, not suitable for constrained devices either.

III. MOTIVATION AND REQUIREMENTS

The main goal of the research presented in this paper is to enable constrained IoT devices to play the ACS roles of User and Verifier, thereby allowing them to interact autonomously in order to authenticate and demonstrate their credential attributes to any Verifier in a privacy-preserving fashion. At the same time, the solution allows IoT devices to verify other peer devices in a M2M environment, addressing the power and memory constrains that many kind of IoT devices need to face.

Applying ACS in IoT constrained devices enables a new vast range of opportunities and scenarios that have not being addressed so far. For example, in Smart-buildings, controlled access systems like certain chips deployed in doors, where the access control depends on validating a set of attributes in a privacy preserving way, can be benefit from our proposal. An IoT device (e.g. optical head-mounted display (OHMD) acting on behalf of the user may present, in a M2M fashion, a zero-knowledge proof to the terminal that acts as a ACS Verifier to open the door based on certain concealed identity attributes. It avoids querying to an external trust party to verify the attributes, giving trust to the administrator that only authorized personnel can access, whereas providing reliability to the Users that their privacy has not been compromised.

Smart-cities and Mobile scenarios can also benefit from our proposal. Thus, vehicles such as a motorcycles, could authenticate its owner by verifying a proof from her wearables, e.g. an smart-watch. When parked, the bike equipped with our solution could verify that another device trying to interact with it belongs to an accredited person, such as a policeman, and in that case the vehicle would disclose its owner's relevant information, proof of passing technical inspections, etc. But, when the vehicle access to a restricted residential area, it would provide an ACS crypto proof in a privacy-preserving fashion stating that its owner's address is within the area, without revealing any other owner's personal identity (e.g. specific address where she lives). These capabilities could be endowed to the vehicles through inexpensive constrained chips with small batteries in order to function independently of an engine running, allowing other vehicles as bicycles to benefit from this privacy preserving solution.

In general, with our proposal, opportunistic, sporadic and pervasive interactions among disparate IoT constrained and autonomous devices will be achieved in a secure, privacy-preserving and M2M fashion. IoT devices will be able to unveil certain data and offer their IoT services to another IoT constrained devices passing by, as long as they satisfy certain identity attributes demonstrated with their ACS attribute credentials.

The requirements of our solution, and therefore or main objectives, are structured below:

- *Security*: the solution should not compromise the security of the device's identities, providing confidentiality and authenticity.
- *Privacy*: the solution should be privacy-preserving, complying with the principle of minimal disclosure of personal information.
- *Usability*: it should provide an API for developers to use the P2ABCE privacy capabilities in their applications.
- *Full P2ABCE support*: the IoT device should be capable of performing every action a traditional ACS roles User or Verifier could perform.
- *Transparent interactions*: any third party actor of the system should not be aware of the IoT condition of the device, therefore, there should be no changes to the P2ABCE protocol.
- *Constrained devices*: the solution should aim to be portable and applicable to as many IoT devices as possible, including those with less computing capabilities.

IV. DESIGN OF THE PROPOSED SOLUTION

To address the aforementioned objectives, and considering the evolution of ACS (and in particular Identity Mixer) to use smart cards, our solution consists on a mandatory implementation of the smart card logic inside the IoT device,

[2]The IRMA project has been recently included in the Privacy by Design Foundation: https://privacybydesign.foundation/

[3]https://github.com/p2abcengine/p2abcengine

[4]Vehicular Ad-Hoc Network

which will conceal all the operations regarding secret keys, and to manage the P2ABCE language, the solution shall offer an API which, at the time being, will perform *computation offloading* to a device capable of running the framework.

Even in the case all P2ABCE were to be implemented inside an IoT device, it should implement the support for software smart cards, to keep the secret inside the IoT device. Therefore, the first step is to implement the smart card logic inside the IoT device, and then, if the device resources admit it, other components of the P2ABCE framework.

Computation offloading is not a new technique for IoT environments. For example, to reduce the overhead of IPv6, 6LoWPAN compresses packets and uses smaller address sizes. In order to communicate a 6LoWPAN with other networks, the IoT devices delegate on a proxy that can manage the 6LoWPAN and IPv6 stacks. In the scope of consumer devices, smart watches can install applications which delegate on the user's phone to accomplish performance demanding task.

Therefore, the IoT device now has a **duality** in its functions, because it is the User that starts any interaction with other actors, and it is also the smart card that a P2ABCE server must ask for cryptographic operations. It can also be seen as a **double delegation**. The IoT device delegates on the external P2ABCE server to manage the protocol, and that P2ABCE server delegates on the IoT device, acting now as the smart card.

Regarding the communication between different P2ABCE actors, it depends on each IoT scenario, independently of our solution, as third party actors have no need to adapt the P2ABCE protocol.

To ensure that our solution does not risk the security of the device's secrets, the next section introduces the fundamentals on the operations performed by a User, therefore identifying the key operations to implement in the smart card.

A. FUNDAMENTALS ON ZERO-KNOWLEDGE PROOFS

To understand Zero-Knowledge Proofs, *Fundamentals of Computer Security* [18, Chapter 12] offers a good introduction to the topic. Here is shown how one can indeed proof knowledge of a value, without revealing it, using the classic ZKP of knowledge based on the discrete logarithm problem, also known as Schnorr's identification scheme. Based on this discrete logarithm ZKP, Idemix then derives multiple ZKPs of relations and properties of the values hidden in a credential.

Using the notation introduced by Camenisch and Stadler [12], the discrete logarithm ZKP can be written as

$$PK\{(\alpha) : y = g^{\alpha}\}$$

given a known group $G = \langle g \rangle$ of prime order q and public value $y \in G$. The notation means: "I know a secret value α such that g^{α} is y", i.e. the discrete logarithm of y, $log_g y = \alpha$.

During the first step of the protocol, the prover chooses randomly a value r, computes the *commitment* $t := g^r$ and sends t to the verifier. Then, the verifier chooses another random exponent, the *challenge* c, and sends it to the prover. Next, the prover computes the *response* $s := r - c\alpha \bmod q$ and sends it to the verifier. Finally, the verifier checks whether or not $t \stackrel{?}{=} g^s y^c$ holds. The verifier never receives the secret value α, and the verifier will not be able to compute it given t and s.

The protocol holds because $g^s y^c = g^{r-c\alpha} g^{\alpha c} = g^r = t$.

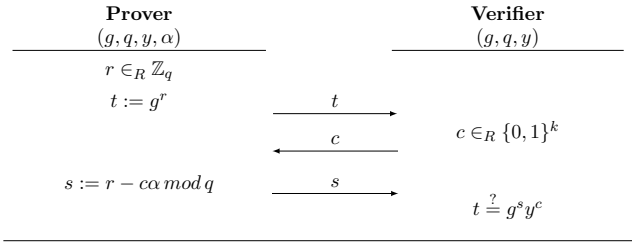

FIGURE 1. Schnorr's protocol $PK\{(\alpha) : y = g^{\alpha}\}$. The prover knows (g, q, y, α) such that $g^{\alpha} = y$. The verifier knows (g, q, y).

The Fiat-Shamir heuristic lets us replace the verifier challenge with a hash function $\mathcal{H}$, computing the challenge as $c := \mathcal{H}(g \mid y \mid t \mid n)$. The value n is a random nonce generated by the verifier at the beginning of the non-interactive ZKP. The nonce makes the verifier trust that the current ZKP is fresh and not a forgery from a previous ZKP.

In Idemix, every ZKP follows the scheme shown above. First, we have the *commitment* `t`, next the *challenge* `c` (computed with $\mathcal{H}$), and finally, the *response* `s`. A prover can achieve parallel ZKPs if she first computes the multiple `t`-values of each individual proof, next, she computes the same challenge `c` for all the proofs by combining all the information in one hash call, and then she computes all the `s`-values in parallel.

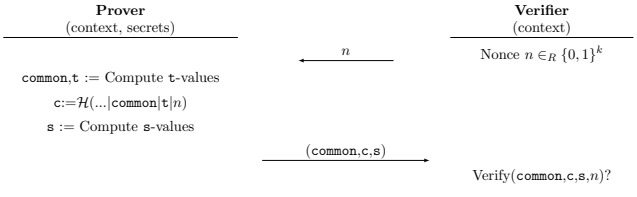

FIGURE 2. Identity Mixer's Non-Interactive ZKP with Nonce.

Fig. 2 shows the Idemix Proving scheme, where prover and verifier share a context information, the verifier generates the nonce, and the prover, non-interactively, computes the proof. The `common` values are equivalent to Schnorr's t, which was shared between prover and verifier. The verifier can recover some $\hat{\texttt{t}}$-values from `common`, `c` and the `s`-values. This is equivalent to recovering Schnorr's t from $g^s y^c$. The verifier then checks whether or not `c` equals $\mathcal{H}(...|\texttt{common}|\hat{\texttt{t}}|n)$ to verify if the $\hat{\texttt{t}}$-values are actually the `t`-values, making the proof valid.

Therefore, to describe any Idemix proof being performed by the User, one can focus on the key steps, compute the `t`-values, the challenge `c` and the `s`-values, and ignore the mathematical operations performed for each specific ZKP.

B. SYSTEM ARCHITECTURE

The system is compounded by the IoT device, the P2ABCE delegation server and the third party P2ABCE actors:

IoT device:

Figure 3 shows our proposed architecture, in which the IoT device is represented with two communication interfaces. One allows external communications to other machines, including other P2ABCE actors. Through this interface, the P2ABCE XML messages are exchanged as in any traditional P2ABCE scenario. This lets an IoT device to interact with other actors without adaptations of the protocol. The other interface allows a secure communication channel with the delegation server. Both, the delegation messages and the APDU Dialogue are transmitted over this interface, making it a point of attack that must be thoroughly secured.

The scheme also shows the *P2ABCE IoT Toolkit*. This piece of software includes the *IoT Smart Card*, and the P2ABCE API.

The *IoT Smart Card* is the implementation of a software smart card, which listens for APDU Commands from the secure interface and stores the credentials and private keys within the device's memory. These secret keys should be kept securely protected inside the IoT smart card. There exist different software solutions, as well as hardware, like Atmel's chips, which offer secure memory and cryptographic operations, with serial interfaces for the IoT devices. Those chips also allow to speed up the SHA256 and AES128 calculations when compared with the performance obtained using software implementations.

The P2ABCE API is an interface for other processes that wish to use the private-preserving environment of P2ABCE. It provides access to every operation available, hiding the delegation process to the server. In the future, for example, the Verification Service could be implemented to run entirely in the IoT device, then the toolkit would conceal the transition from delegation to native execution.

P2ABCE actors:

The possible roles in a P2ABC system are the Issuer, the User, the Verifier, the Revocation Authority and the Inspector, where these last two actors are optional. All of them use the P2ABCE language to communicate to each other. Any third party actor that communicates with the IoT device will be unaware of the fact that the device is a constrained device, because it will accept and generate the same XML as a traditional User.

Figure 4 showcases the different actors and their interactions, where the User is in the center, which receives a credential from an Issuer, can generate privacy-preserving Tokens for Verifiers or revoke an stolen credential. The Inspector is the only entity capable of reading some ciphered attributes from a Token, if the User accepted to include that ciphered information in it. It serves the Law Enforcement authorities, warrant granted, to track any misuse of a credential.

P2ABCE Delegation Server:

The machine in charge of receiving commands from authorized IoT devices to parse the XML files exchanged. It will also orchestrate through APDU Commands the cryptographic operations the IoT smart card must perform.

C. DELEGATION PROCESS

This section describes the computation offloading carried out by the IoT device. The steps the device will perform in any kind of interaction are:

- Communication with P2ABCE actor
  The IoT device, acting as a User, starts an interaction with another actor. If it is an Issuer, it will start the issuance process. If it is a Verifier, then it will provide a Presentation Policy for the device to proof in a privacy preserving way. Figure 5 shows an example of this last case.
  It may also happen that the device is contacted as a Verifier by another actor, e.g. in a M2M scenario where the IoT device requires authentication to access to its resources.
- Delegation to the P2ABCE Server
  Depending on what role the IoT device is acting as, it will use the corresponding API from the P2ABCE IoT Toolkit. The selected API will delegate to the Service deployed in the delegation server, e.g. User Service. The delegation message will include the XML data, and any parameter required to accomplish the task, as the information on how to communicate back with the IoT smart card.
  The server will then parse the XML messages and begin to orchestrate the response. In the case of the Verifier Service, it will answer immediately with an *accept* or *reject* of the Token. In case it is the User Service, it will need information from the IoT credentials and cryptographic operations.
- APDU Dialogue
  Through the secure channel between IoT device and server, the User Service will send APDU Commands to the *IoT smart card* to read the credential information or perform cryptographic operations involving private keys, necessarily stored inside the IoT device.
  During a proof, the Service will read the credential public information to fill the Presentation Token, and then will request the IoT smart card to calculate the `t`-values of the proof. The Service will read the `common` and `t`-values, use the nonce present in the Presentation Policy of the Verifier and compute the challenge `c`. Next, the Service will send `c` to the IoT smart card through another APDU Command to request the calculation of the `s`-values.
  After the APDU Dialogue, the Service has all the needed values to fill in the Token, and the IoT device

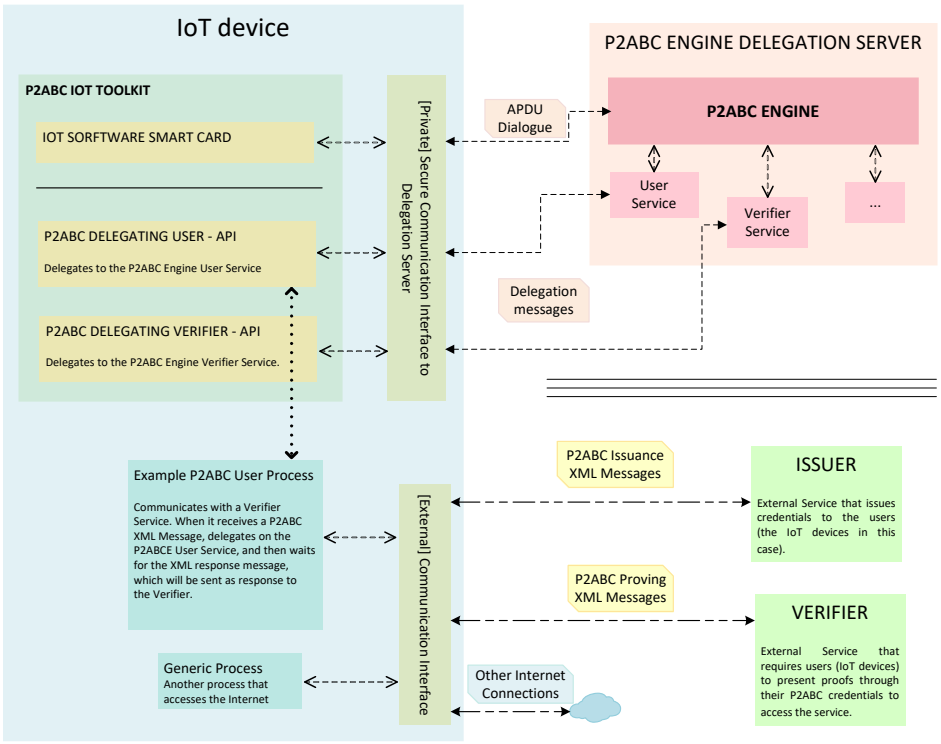

FIGURE 3. Proposed high level Architecture for integrating IoT devices in P2ABCE.

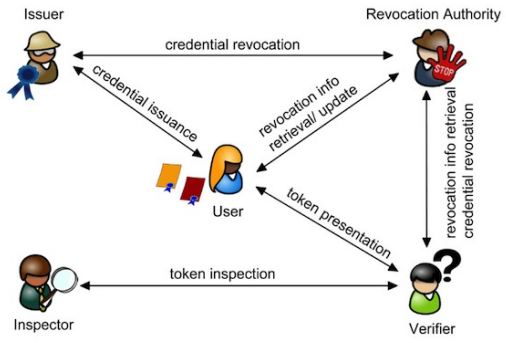

FIGURE 4. Entities in a P2ABC System. Source: P2ABCE project.

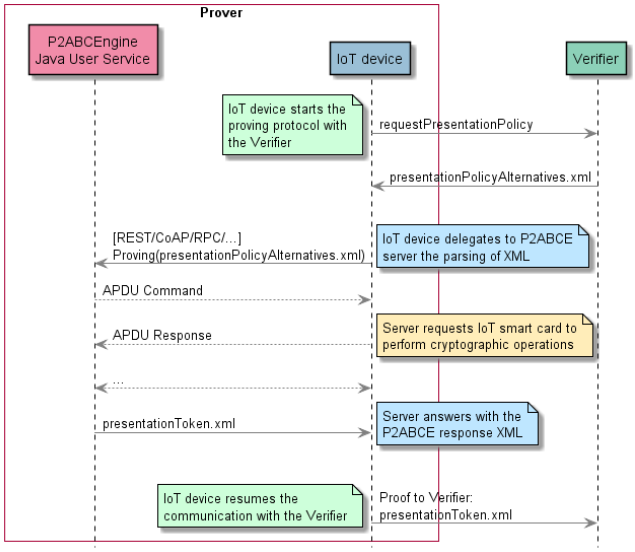

FIGURE 5. Messages exchanged during the proving delegation.

performed all the operations involving private values on its own.

- Server response
  After the APDU Dialogue, if needed, the server may return a status code indicating success or failure, or a XML response if the third party actor requires an answer from the IoT device, as a Presentation Token, or the intermediate issuance messages.

The *Server-IoT* channel must be secured. It must avoid impersonation of the P2ABCE delegation server or authorized devices. For this purpose many traditional solutions already exist. The delegation server could be in a local network, or physically attached to the IoT device, like the Arduino Yún[5] combines an ATmega and an Atheros with Linux. The choice may depend on the particular deployment.

[5]https://www.arduino.cc/en/Main/ArduinoBoardYun

V. IMPLEMENTATION

This section presents the first Proof of Concept (PoC) implementation, introducing the IoT system used in the tests, the delegation protocols, one for the computation offloading of the IoT device on the P2ABCE server, and another one for

the transmission of APDU Commands, and finally, the IoT smart card implementation.

The PoC is developed under a Linux based system aimed for IoT environments, a fork of OpenWrt, called LEDE (Linux Embedded Development Environment), that will serve as a starting point for future implementations in more constrained devices. The delegation server will run on Raspbian OS, in a Raspberry Pi 3, with a suitable Java Runtime for P2ABCE to run.

After the device receives a P2ABCE message from a third party actor, the delegation process begins. It consists on two steps, first the IoT device calls the P2ABCE server to offload the parsing of the XML data, and then the P2ABCE server, sending APDU Commands, delegates on the IoT smart card the cryptographic calculations, if needed.

A. POC DELEGATION TO P2ABCE SERVER

Currently the P2ABCE project offers multiple REST web services to run different roles in the P2ABCE system: User Service, Issuer Service, Verification Service, etc. These services make use of helper libraries that can be used to implement other services, with more suitable protocols for each specific deployment.

After an analysis of the project's code, the smart card logic was found in the `Smartcard` interface and its implementations, `HardwareSmartcard` and `SoftwareSmartcard` classes. The first one uses the `javax.smartcardio` abstract classes to communicate with the physical smart cards. The Oracle JRE implements these package for the majority of smart card manufacturers. For our PoC, `javax.smartcardio` package was implemented, as `IoTsmartcardio`, so it transmits the APDU Commands and Responses with a simple custom protocol, making the use of a physical or IoT smart cards totally transparent to the `HardwareSmartcard` class, enhancing maintainability, and following the *expert pattern* from the known GRASP guidelines.

In this PoC, the P2ABCE's User Service was modified to add a new method receiving the IoT device's IP address and a port where the IoT Smart Card will be listening for the APDU Commands, the information needed for the custom protocol to transmit APDUs. The remaining REST methods of the services are left untouched.

B. APDU DIALOGUE TRANSMISSION

The transmission of APDU messages between the delegation server and the IoT device is done through a custom simple protocol, referred as BIOSC (Basic Input Output Smart Card).

It consists on one first byte for the instruction, that can mean either an APDU Commnad or a finishing instruction to close the connection. In the first case, the header continuous with two more bytes for the length of the payload bytes to read, which are the APDU Command bytes. To send an APDU Response in BIOSC, the IoT device sends back to the server two bytes for the length and then the raw APDU Response bytes. The messages are sent over TCP for a reliable transmission.

The protocol lacks any security (authentication or authorization) that a real system should implement. It is vital to authenticate the delegation service, to authorize it to perform APDU Commands, and the same goes for the IoT device, to prevent impersonation attacks. But using BIOSC for the transmission of the APDU messages in the PoC, with only 3 bytes of overhead, will help in the benchmarks to measure the performance of the system, independently of the connection to the delegation service.

C. IOT SMART CARD IMPLEMENTATION

This section showcases the code architecture and sequence diagrams of the IoT Smart Card PoC.

Figure 6 shows how the project is divided in three different sections, with the objective of enhancing maintainability, improving future changes, ports and error fixes.

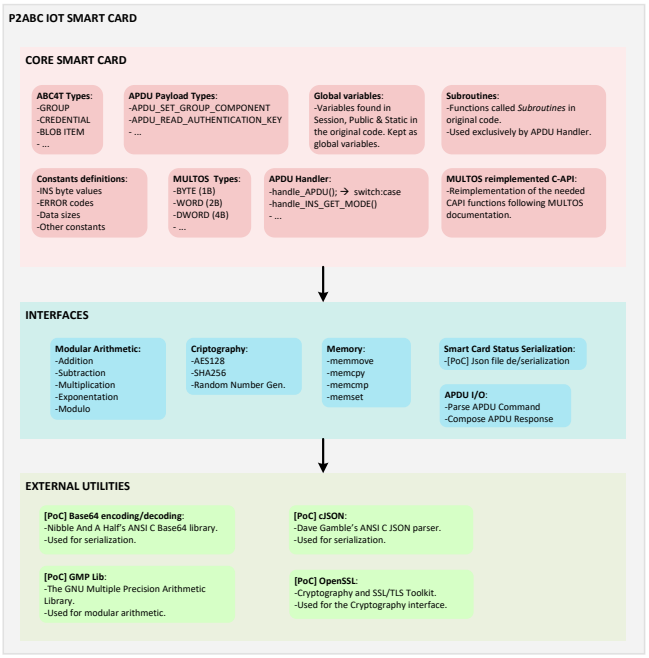

FIGURE 6. IoT Smart Card Code Structure.

Core smart card: The smart card logic lies here, with the concepts of APDU Commands and Responses, the instructions that are defined for P2ABCE smart cards, and how to process them and perform the cryptographic calculations.

It is mainly based on ABC4Trust Card Lite's code, a MULTOS physical smart card implementation of P2ABCE smart cards, but also on the `SoftwareSmartcard` Java implementation for a high level reference. MULTOS acts as an operative system for smart card applications. It provides a custom API for memory management, arithmetic functions, etc. It also uses an architecture different from traditional computers. This implied an almost integral reimplementation of the original code.

In some cases, to avoid some MULTOS API limitations, like in [23] where they noted that MULTOS' `ModularExponentiation` did not accept exponents larger than the modulus size, an equivalent function with *expanded functionality* was implemented.

Another notable difference is that MULTOS compiler does not apply data structure alignment. This affects the inherited ABC4Trust's code because of the massive use of `memcpy` to copy multiple adjacent variables with one function call, usually when reading or writing APDUs. A temporal solution is to use `struct __attribute__((__packed__))` to ask the GCC compiler not to use padding in the structs, but this is not standard functionality, neither a good practice. A deeper refactorization of the code would be needed where the obscured copies of variables must be made explicit, letting the compiler manage the memory layout on its own.

Interfaces: To reimplement some of the MULTOS functions, a facade isolates the implementation of the core smart card from auxiliary function implementations, that could vary depending on the hardware or the system used by the IoT device. With this facade, for example, it could be possible to change the implementation of cryptographic functions to use hardware acceleration (e.g. Atmel's chips for SHA and AES), or new software implementations optimised for the target platform.

The interfaces defined can be organized in 5 groups (see Fig. 6), depending on their purpose: Modular Arithmetic, Cryptography, Memory Management, Serialization and APDU Parsing.

External utilities: To implement some *interfaces*, the current PoC uses two ANSI C libraries, for base64 and JSON, and two shared libraries available as packages in LEDE: GMPLib and OpenSSL. These libraries use dynamic memory and offer more functionality than what is actually needed, and although they are useful tools for the early development versions of the PoC, future versions should replace them for more lightweight solutions.

The *interfaces* and *external utilities* sections allow that the project is easily ported to specific targets without modifying the smart card logic.

The sequence diagram from Figure 7 shows the execution of the PoC IoT smart card.

The execution starts with the `main` function in `BIOSC.c`, first it deserializes the current smart card status from a Json file (which helps with debugging the IoT smart card), next, it listens on a loop for APDU Commands from the delegation server.

Every time an APDU Command arrives, it calls the function `handle_APDU()` with the raw APDU bytes. The Handler calls the APDU I/O interface to parse the bytes, storing in different variables the APDU structure. Using a `switch-case` expression on the `INS` byte, the Handler calls a fitting *Instruction handler* function.

Inside this function, it may call multiple functions from the Subroutines, that may call MULTOS C-API functions, which, in turn, may use an interface to perform its functionality, depending on the specific instruction being handled.

Finally, every instruction handler must end. Right before the `return;` expression, they must call `mExit`. This reimplemented MULTOS function will save the current status of the smart card and send the formed APDU Response to the delegation server.

After returning from the APDU handler, the program listens again from the socket, until a BIOSC finishing instruction arrives.

VI. VALIDATION AND PERFORMANCE EVALUATION

This section describes the deployment of three testing scenarios with different hardware capabilities. A laptop, a standalone Raspberry Pi 3 (RPi3), and an Omega2 IoT device with the Raspberry Pi 3 as the delegation server. The scenarios aim to measure and compare the results to determine whether the proposed solution runs as expected and it is feasible.

A. TESTBED DESCRIPTION

To verify the correct execution of the *IoT smart card*, the test is based on the ABC system from the tutorial in the P2ABCE Wiki[6]. It consists on a soccer club, which wishes to issue VIP-tickets for a match. The VIP-member number in the ticket is inspectable for a lottery, ie. after the game, a random presentation token is inspected and the winning member is notified.

First, the **setup** phase takes place, where several artifacts are generated and distributed to the various P2ABCE entities. Next, a ticket credential containing the following attributes is **issued**:

```
First name: John          Member number: 23784638726
Last name: Dow            Matchday: 2013-08-07Z
Birthday: 1985-05-05Z
```

Then a **presentation** is performed, where a *presentation policy* from the verifier specifies that the member number is inspectable and a predicate which ensures that the matchday is in fact $2013-08-07Z$. This last part ensures that a ticket issued for another match can not be used. The policy also includes the nonce value for the proof's challenge.

First we will execute the test in our development laptop. After asserting that the services work as expected, we then proceed to run the test in a RPi3[7], exactly like in the laptop. Finally, the PoC IoT smart card will be deployed in an Omega2[8] and the delegation services in the RPi3.

[6]https://github.com/p2abcengine/p2abcengine/wiki/
[7]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[8]https://docs.onion.io/omega2-docs/omega2.html

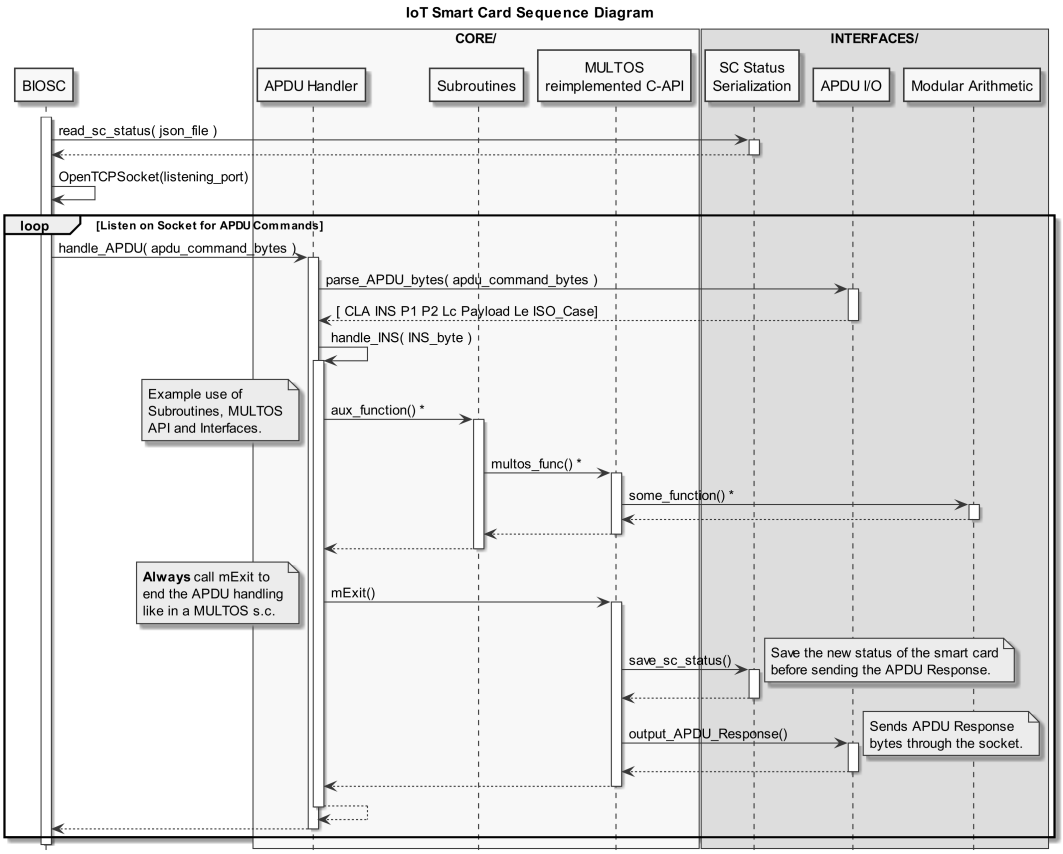

FIGURE 7. IoT Smart Card Sequence Diagram.

The third scenario involves communications over TCP in a LAN. The RPi3 is connected over Ethernet to a switch with a WiFi access point. The Omega2 is connected over WiFi n to said AP. To test the possible network delays, 6000 samples of APDU messages were taken from the test executions, and sent between the two devices, without handling the APDUs. The results show a mean of $14\mu s$ per byte sent, i.e. less than half a millisecond per APDU.

B. RESULTS

After 20 executions for each scenario (laptop, RPi3, Omega2+RPi3), the time was measured for each REST call executed against the User Service in the delegation server, to later take the means to compare each step of the testbed. Because during a call to the User Service, the server may contact the *IoT Smart Card*, the difference in times between the second and third scenario will show the performance of our IoT PoC.

It is worth noting that during the tests, the CPU usage showed that P2ABCE does not benefit from parallelization, therefore it only uses one of the four available cores in the laptop and Raspberry Pi 3.

The IoT device does not intervene in the first step of the testbed, the setup, until the initialization of the new smart card, thus the times measured for each REST call in the second and third scenarios are practically identical as shown in Fig. 8. The laptop is about ten times faster than the Raspberry Pi 3. It is also noticeable that the setup step is only done once, and the highest time measured is less than two and a half seconds.

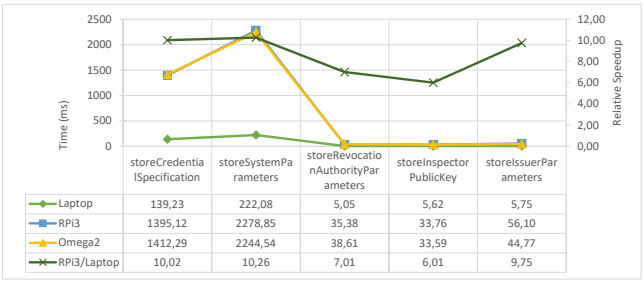

FIGURE 8. Setup times (milliseconds) and relative speedup.

The creation of a `SoftwareSmartcard` or a `HardwareSmartcard`, which actually uses the `IoT Smart Card`, differ in how the hardware version must copy to an external device the cryptographic information of the system, shared during the setup phase, as well as common smart card functionality, like the PIN, PUK, operation mode,

etc. Figure 9 shows this difference between the software versions of the laptop and RPi3, and the IoT version. This is a one time procedure per device.

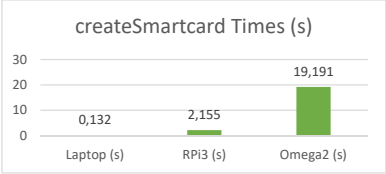

FIGURE 9. Create smart card times (seconds).

As this is the first interaction between the RPi3 and the IoT smart card running in the Omega2, it is interesting to analyse the possible network delays. The process uses 30 APDU Commands, and their respective Responses, with a total of 1109 bytes. From the network benchmark, the delay in the transmission is around 15 and 20 ms, negligible compared to the total operation time.

The results for the Issuance step of the credential with 5 attributes and key sizes of 1024 bits is shown in Figure 10. The process is performed with three REST calls to the User Service, corresponding to the multi-step Idemix credential issuance and identity to use selection. The results shows the times for each call in each device, and the increase of time from the RPi3 to the Omega2 case highlights the difference of processing power between the devices, where the first one makes use of the Java implemented smart card, and the Omega 2 runs the IoT PoC smart card.

The three REST calls to the delegation service involved APDU Dialogues with the IoT smart card, with 45 APDU Commands in total, 3197 bytes exchanged, that would have introduced a latency of only 45ms from the network.

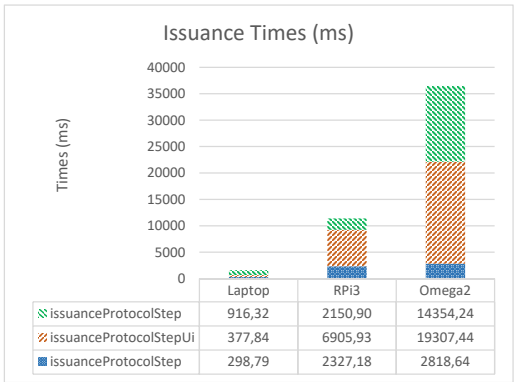

FIGURE 10. Issuance times (milliseconds)

The final step of the test involves a Proving or Presentation in P2ABCE, where the Verifier sends the User a Presentation Policy, and the User answers with the Presentation Token, without more steps from the User. To ensure that all the process was successful, it was enough to check from the Verifier and the Inspector perspectives that every Token generated was valid.

As shown in Fig. 11, there is a correlation between the cryptographic work the IoT smart card must perform, and the increase of time measured from the RPi3 to the Omega2 cases. The APDU Dialogue involved 28 APDUs, with 1939 bytes, about only 27ms of delay from the network.

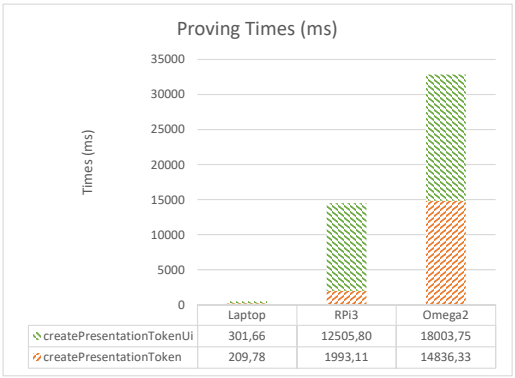

FIGURE 11. Proving times (milliseconds)

Unlike the previous steps, the Proving is the most used method and key feature of the P2ABCE ecosystem. The laptop performs a prove in less than one second, the RPi3 standalone needs 15 seconds, but the P2ABCE IoT deployment needs almost 15 seconds for the first step, and 18s for the second step, 33 seconds total to generate a Presentation Token.

After the tests, in a new series of executions, the tool `time -v` provided the `Maximum resident set size (kbytes)` of the PoC IoT smart card, i.e. the maximum size of RAM used by the process since its launch. The mean of the maximum memory usage measured was 6569.6 kbytes. This includes the use of static memory for all the *global variables* of the smart card logic, as well as the dynamic memory allocated by the external utilities libraries, GMPlib, OpenSSL and cJSON.

GMP and OpenSSL always allocate the data in their own ADT, which involves copying the arrays of bytes representing the big modular integers from the cryptographic operations instead of using the byte array representations of the smart card logic. cJSON, used in the serialization of the smart card, for storage and debugging purposes, stores a copy of every saved variable in a JSON tree structure, then allocates a string with the JSON, that then the user can write to a file. These multiple poor uses of memory could be avoided in future versions of the PoC, with a custom binary serialization or arithmetic library.

C. VALIDATION CONCLUSIONS

Table 1 sums up the time used in each step of the test for the IoT scenario. The first step, *System Setup* is done only once when the system is being deployed, and the *IoT Smart Card Setup* only once per device. Because a device can have more

than one credential, the Issuance step is significant when we issue multiple credentials over the device's lifetime. Finally, the Proving step is expected to be the most commonly performed operation. The times obtained basically concur with the tests performed in [4] for their smart card implementation, where the bottleneck is the copy of group parameters and other data from the IoT smart card to the server. The rest of the time is spent mostly with the cryptographic operations. The fact that it lasts over half a minute implies that we should not use this PoC for *real-time* applications yet. Nevertheless, for many other IoT applications, the fact this operation can be performed multiple times per hour, presents an useful tool for privacy.

System Setup	IoT Smart Card Setup	Credential Issuance	Prove Presentation Policy
3.77 s	19.19 s	36.48 s	32.84 s

TABLE 1. Total time spent for each step in the Omega2+RPi3 scenario.

VII. CONCLUSIONS AND FUTURE WORK

This paper has presented a generic privacy-preserving solution for the vast world of the Internet of Things. The IoT devices can operate as individual actors in the P2ABCE ecosystem, and when in need of performing computation offloading, the delegation server can also be a device considered into the IoT family. Out PoC implementation demonstrates the feasibility of the proposal to integrate ACS in IoT constrained scenarios. Unlike in current state of the art proposals that performs simulation of IoT environments and the scenarios are not actually constrained (e.g. in [15] or [2], our approach has been implemented, deployed and validated in a real IoT scenario with constrained hardware IoT devices.

As future work, we envisage to study the available solutions for delegation, e.g. using REST, CoAP, RPC, and for the security issues mentioned, in order to address the definition of the P2ABCE API to abstract the delegation process to other processes running in the IoT devices. We also expect to continue evolving the implementation, where it would be interesting to compare the execution of the current PoC to a version with cryptographic hardware acceleration, for example, using Atmel's chips for SHA, AES and secure memory.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] Specification of the identity mixer cryptographic library (v2.3.43). Technical report, IBM Research, January 2013.
[2] Almudena Alcaide, Esther Palomar, José Montero-Castillo, and Arturo Ribagorda. Anonymous authentication for privacy-preserving iot target-driven applications. Computers & Security, 37:111–123, 2013.
[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. Computer Networks, 54(15):2787 – 2805, 2010. http://www.sciencedirect.com/science/article/pii/S1389128610001568.
[4] Thomas Baignères, Patrik Bichsel, Robert R. Enderlein, Hans Knudsen, Kasper Damgård, Jonas Jensen, Gregory Neven, Janus Nielsen, Pascal Paillier, and Michael Stausholm. D4.2 final reference implementation. Technical report, ABC4-Trust, August 2014. https://abc4trust.eu/download/D4.2
[5] Jorge Bernal Bernabé, José Luis Hernández Ramos, and Antonio Fernandez Gómez-Skarmeta. Holistic privacy-preserving identity management system for the internet of things. Mobile Information Systems, 2017:6384186:1–6384186:20, 2017.
[6] P. Bichsel, J. Camenisch, T. Grob, and V. Shoup. Anonymous credentials on a standard java card. ccs, 2009.
[7] S. Brands and C. Paquin. U-prove cryptographic specification v1.0. tech. rep. Technical report, Microsoft, March 2010.
[8] S.A. Brands. Rethinking public key infrastructures and digital certificates: Building in privacy. mit press, August 2000.
[9] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Advances in Cryptology—EUROCRYPT 2001, pages 93–118. Springer, 2001.
[10] Jan Camenisch and Anna Lysyanskaya. An eficient system for non-transferable anonymous credentials with optional anonymity revocation. Advances in Cryptology EUROCRYPT 2001, 2001.
[11] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In International Conference on Security in Communication Networks, pages 268–289. Springer, 2002.
[12] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. Advances in Cryptology—CRYPTO'97, pages 410–424, 1997.
[13] Jan Camenisch and Markus Stadler. Eficient group signature schemes for large groups. Advances in Cryptology CRYPTO 97, 1997.
[14] Luuk Danes. Smart card integration in the pseudonym system idemix. Master's thesis, Faculty of Mathematics. University of Groningen, 2007.
[15] J. M. de Fuentes; L. González-Manzano; J. Serna-Olvera and F. Veseli. Assessment of attribute-based credentials for privacy-preserving road traffic services in smart cities. Personal and Ubiquitous Computing Journal, Special Issue on Security and Privacy for Smart Cities, February 2017. http://www.seg.inf.uc3m.es/ jfuentes/papers/PrivacyABC-VANET.pdf.
[16] Tom Henriksson. How strong anonymity will finally fix the privacy problem. VentureBeat, October 2016. https://venturebeat.com/2016/10/08/how-strong-anonymity-will-finally-fix-the-privacy-problem/.
[17] Víctor Sucasas Iglesias. Implementation of an anonymous credential protocol. Master's thesis, Escuela Técnica Superior de Ingenieros de Telecomunicación. Universidad de Vigo, 2008-2009.
[18] Jennifer Seberry Josef Pieprzyk, Thomas Hardjono. Fundamentals of Computer Security. Springer, 2003.
[19] Gregory Neven. A quick introduction to anonymous credentials, August 2008.
[20] Giuseppe Persiano and Ivan Visconti. An Efficient and Usable Multi-show Non-transferable Anonymous Credential System, pages 196–211. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
[21] Zhiyun Qian, Z. Morley Mao, Ammar Rayes, David Jaffe (auth.), Muttukrishnan Rajarajan, Fred Piper, Haining Wang, and George Kesidis (eds.). Security and Privacy in Communication Networks: 7th International ICST Conference, SecureComm 2011, London, UK, September 7-9, 2011, Revised Selected Papers. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 96. Springer-Verlag Berlin Heidelberg, 1 edition, 2012.
[22] M. Sterckx, B. Gierlichs, B. Preneel, and I. Verbauwhede. Efficient implementation of anonymous credentials on java card smart cards. in: Wifs, 2009.
[23] Pim Vullers and Gergely Alpar. Efficient selective disclosure on smart cards using idemix. In IFIP Working Conference on Policies and Research in Identity Management, pages 53–67. Springer, 2013.