Universitat de les Illes Balears

escola politècnica superior

Treball Final de Grau

Disseny i Implementació d'un sistema de gestió de Targetes Turístiques Ciutadanes

SEBASTIÀ GALMÉS ALBA

Tutor
Maria Magdalena Payeras Capellà

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 19 de juny de 2017

SUMARI

ACRÒNIMS

ADS Autonomous Decentralized Systems

NFC Near Field Communication

TTP Trusted Third Party

ZKP Zero Knowledge Proof

WAR Web application ARchive

API Application Programming Interface

ORM Object-relational Mapping

SO Sistema Operatiu

AP Acces Point

URL Uniform Resource Locator

HTTP HyperText Transfer Protocol

CA Certificate Authority

JSON JavaScript Object Notation

WLAN Wireless Local Area Network

RESUM

El desenvolupament exponencial que han viscut els dispositius mòbils, juntament amb l'augment del turisme, han suposat un interès creixent en la creació d'un sistema per a integrar les noves capacitats d'aquests dispositius per a millorar l'experiència dels turistes. L'alt cost de desenvolupar i, sobretot, desplegar un sistema d'aquestes característiques ha provocat que cap de les solucions proposades hagi pogut establir-se com un nou estàndard per a oferir aquests serveis.

En aquest document s'explicarà una nova proposta creada per Payeras-Capellà et. al a l'article "*mCITYPASS : Privacy-preserving Secure Access to Federated Touristic Services with Mobile Devices*"[1], on es presenta un sistema federat què garanteix l'anonimat reversible dels usuaris, alhora que permet oferir diversos tipus de serveis segons el nombre de vegades que s'hi pot accedir.

A continuació es mostrarà la implementació d'aquest sistema amb els objectius de demostrar què aquest protocol és implementable i que compleix amb els requeriments temporals d'una plataforma d'aquestes característiques. A més a més, es mostrarà el desenvolupament d'un prototip per a mostrar les interaccions dels usuaris amb aquest sistema.

CAPÍTOL 1

INTRODUCIÓ

Les Targetes Turístiques Ciutadanes són targetes que permeten la integració dels diversos serveis turístics i ofertes que ofereix una ciutat a una mateixa plataforma, especialment les atraccions turístiques i els sistemes de transport. El seu objectiu és combinar els diferents serveis a una simple plataforma per a millorar el servei de qualitat dels usuaris del transport públic per a convèncer als usuaris sobre l'ús d'alternatives al cotxe [2].

A més a més, aquesta plataforma pot ser estesa a llocs turístics i d'entreteniment per a facilitar l'accés a punts d'interès de les ciutats fent aquest servei interessant per a turistes. Moltes ciutats al voltant del món ja ofereixen targetes que permeten l'ús del transport públic així com un descompte o entrada gratuïta a punts d'interès. [2].

La principal dificultat alhora de posar una targeta d'aquestes característiques és aconseguir la implicació necessària de tots els interessats necessaris per a obtenir la millor experiència possible. Així, la presència de diferents operadors de transport públic i gestors d'atraccions turístiques (ja siguin públics o privats) així com els operadors dels altres serveis oferits poden dificultar la implementació d'una plataforma amb la interoperabilitat necessària per a que l'usuari percebi un benefici real amb l'ús d'aquest sistema.

La tendència principal d'aquest tipus de plataforma ha estat la implementació de sistemes basats en *smart ticketing*. D'acord amb l' URBAN ITS EXEPERT GROUP, *smart ticketing* és üna eina per a simplificar l'accés al transport públic per a l'usuari final i millorar la mobilitat sostenible"[3]. L'objectiu d'aquesta tecnologia és simplificar el sistema de pagament als usuaris, oferint diversos mètodes i processos de pagament alhora que s'assegura als operadors que es cobraran les tarifes de manera efectiva. A més a més, aquests sistemes incorporen una funció estadística que permet als operadors de transport públic millorar el seu servei recopilant informació sobre els trajectes dels usuaris [3].

A continuació s'estudiarà alguna de les targetes existents i les prestacions que ofereixen.

1.1 Oyster Card i London Pass, Londres

Londres és una de les ciutat més grans d'Europa, un dels principals centres econòmics del món, així com un destí turístic que acull més de 65 milions de persones [4]. Per a fer front a tal demanda, es realitzaren diversos intents durant els anys 90 d'implementar un sistema de targetes intel·ligents. Tot i que aquests sistemes no varen triomfar, varen proporcionar valuosa informació sobre les característiques que hauria de tenir aquest tipus de sistemes per a funcionar correctament. Especialment varen mostrar que era necessari que la cooperació total entre els diferents *stakeholders* i especificacions ben definides [2].

Amb aquests objectius, el 2003 es va introduir l'Oyster Card amb una acceptació excel·lent des del principi. Aquesta targeta funciona a tot el sistema de busos i metros de Londres, algunes barques i trens lleugers i trens d'àmbit nacional que arriben o parteixen de Londres. A més a més, ofereix diferents tipus de facturació segons les necessitats de l'usuari, com per exemple viatges senzills o passos temporals [2].

Enfocat al sector turístic, la ciutat també ofereix el *London Pass* que permet l'entrada a més de 50 atraccions sense la necessitat de realitzar coa per entrar. A més a més, ofereix descomptes a molts restaurants, tendes i altres opcions d'entreteniment. Tot i així, aquesta *destination card* no inclou transport. Així doncs, els turistes disposen de dues opcions per a emprar el transport públic: comprar un *ticket* de paper temporal per a 24h o 7 dies o obtenir una *Visitor Oyster Card*. Aquesta ofereix un servei *pay-as-you-go* però a tarifes molt més baixes [2].

1.2 Octopus Hard, Hong Kong

Hong Kong va començar el seu desenvolupament com a colònia britànica, però vas des de convertir-se en la primera regió administrativa especial de Xina, ha explotat econòmicament i s'ha transformat en una potència econòmica a nivell mundial, així com una de les portes de l'economia xinesa cap al món occidental.

Aquest creixement ha convertit Hong Kong una de les ciutats més densament poblades, fent necessària la implementació d'un transport públic eficient. Per a facilitar el transport, les cinc companyies privades que gestionen el transport públic a Hong Kong ofereixen l'*Octopus Card* des de 1994 [2]. Des de la seva presentació, va ser acceptada per gairebé tota la població local, amb un 90% d'acceptació el 2004 entre la població de 15 a 90 anys. Aquest èxit va suposar que s'afegissin altres serveis com a pagament a tendes, màquines expenedores i fotocopiadores, parquímetres, control d'accés, serveis governamentals i oci [2]. Aquests serveis addicionals suposen la majoria de les transaccions emprant l'*Octopus Card*.

A més a més, Hong Kong és una de les principals destinacions turístiques xineses. Enfocat a aquest sector, la ciutat ofereix una targeta especial per a turistes (*sold tourist octopus card*). Aquesta targeta es pot emprar per a accedir a la majoria d'atraccions de Hong Kong així com per a accedir al transport públic.

1.3 Sant Peteburg City Card, Sant Petesburg

Sant Petesburg és considerada una de les ciutats més emblemàtiques d'Europa, amb parts del centre reconeguda com a patrimoni de la humanitat per l'UNESCO. Acceptada com la capital cultural de Rússia, el nombre de visitants ha augmentat considerablement en els darrers anys. Tot i així la ciutat presenta dos inconvenients considerables per a convertir-se en un referent a nivell mundial: la mala qualitat del seu transport públic i ser una de les ciutats més cares a l'hora de visitar-la [2].

Per a fer front a aquest problema, la ciutat ofereix des del 2013 una targeta de visita enfocada al sector turístic amb entrada gratuïta a la majoria de museus i atraccions de la ciutat, excursions gratuïtes, descomptes a tendes, restaurants i hotels, així com accés al transport públic. Tot i oferir un gran nombre de serveis, la seva implementació és de les manco avançades de totes les ciutats amb targetes d'aquestes característiques. L'entrada als museus es gestionada mitjançant segellant un catàleg que s'entrega a l'usuari i l'accés al transport públic implica la necessitat d'una targeta electrònica addicional.

1.4 EZ-Link, Singapur

Des de la seva independència el 1965, Singapur s'ha transformat d'un dels països manco desenvolupats a una potència econòmica a nivell mundial. Per a mantenir aquest creixement, una de les prioritats del govern va ser la creació d'un sistema de transport públic, juntament amb la integració de la planificació urbanística i del transport.

El 2002 es va introduir el sistema EZ-Link amb l'objectiu de facilitar l'accés, millorar el *throughput* de passatgers i obtenir informació en temps real sobre els patrons de viatge. [2]. El 2009 el sistema es va actualitzar al protocol CEPAS (Contactless ePurse application standard). Aquest protocol té com a objectiu crear una sola targeta de transport alhora que s'obre la porta a la creació de noves aplicacions per terceres parts. A més a més, en els darrers s'ha millorat el sistema oferint la possibilitat d'emprar dispositius mòbils compatibles amb NFC, afegint moltes més funcionalitats. La publicació de l'estàndard també permet a terceres parts crear les seves pròpies targetes, afegint serveis d'*e-Wallet* i *e-Government* i recol·lectar informació sobre els usuaris. [2].

CAPÍTOL 2

ESTAT DE L'ART

En els darrers anys, la majoria de ciutats europees han implementat mesures per a limitar el tràfic de cotxes al centre de les ciutats i promoure l'ús del transport públic. Per respondre a aquesta nova demanda, s'han proposat diverses solucions acadèmiques que ofereixen diferents serveis segons l'objectiu dels autors.

2.1 Estudi de propostes de Targetes Ciutadanes

La recerca sobre les targetes ciutadanes s'ha centrat en tres aspectes diferents:

- Estructura de la xarxa.
- Tecnologies de Comunicació.
- Protocols de Comunicació.

2.1.1 Estructura de la xarxa

L'estructura de xarxa més proposada a la literatura es basa en els sistemes distribuïts(Autonomous Decentralized Systems (ADS)). Aquests sistemes varen ser proposats per primera vegada als anys 70, però la millora en computació i, especialment, comunicacions han permès el desenvolupament de sistemes cada vegada més complexes. ADS ofereix una millora en l'eficiència del cost, productivitat de software, flexibilitat i adaptabilitat [5].

Aquest tipus de sistema és l'emprat pel sistema SUICA desenvolupat per la Companyia de Trens de l'Est del Japó (JR East) per al cobrament de tarifes i *e-commerce*. Aquest sistema es va dissenyar amb la capacitat de millorar el seu funcionament o reparar una de les parts sense aturar l'operació del sistema [5]. Aquests sistemes es caracteritzen perquè el sistema és el resultat de la integració de diversos subsistemes i és normal que es produeixin errors [5]. Els subsistemes es caracteritzen per:

- Uniformitat en l'estructura: Cada subsistema és uniforme en estructura i autònom.
- Localitat en la informació: Cada subsistema es coordina a ell mateix i es coordina només amb altres basat en informació local.
- Igualtat en funció: Cada subsistema és igual en funció.

2.1.2 Tecnologies per a la comunicació

L'augment en l'interès per integrar els sistemes d'*e-ticketing* als *smartphones* ha suposat una lluita entre les tecnologies disponibles per a realitzar la connexió entre els dispositius mòbils i els dispositius per a la verificació dels *tickets*. Les tres tecnologies principals són:

NFC, *Near Field Communication*

Near Field Communication és una tecnologia de comunicació sense fils que permet la comunicació a curta distància entre dispositius compatibles [6]. Tot i que el cost no és molt alt, la tecnologia encara no és molt coneguda i no està disponibles a tots els *smartphones*.

QR, *Quick Response Codes*

Quick Response transmeten la informació emprant codi de barres en dues dimensions, permetent emmagatzemar molta més informació que un codi de barres normal. Tot i ser molt acceptada i barata, el procés de lectura és molt més lent que amb Near Field Communication (NFC).

BLE, *Bluetooth Low Energy*

Bluetooth Low Energy és una nova versió de *Bluetooth* amb un consum menor i un abast major. Aquesta nova tecnologia ha permès l'aparició d'un nou tipus de dispositius anomenats *beacons*. Aquests dispositius envien senyals *Bluetooth* periòdiques que els dispositius que estiguin escoltant poden emprar per a determinar la distància al *beacon* o determinar si el dispositiu s'atraca o s'allunya del *beacon*. Els principals avantatges dels*beacons* és que no necessiten la interacció de l'usuari i permeten una comunicació a molta més distància, però a un cost molt més elevat.

2.1.3 Protocols de comunicació

La major part de la literatura sobre els *tickets* electrònics es basa en els protocols emprats per a realitzar la comunicació. La majoria dels sistemes d'aquestes característiques que s'han arribat a implementar són d'àmbit local i no sempre funcionen entre diferents modes de transport. Aquesta diversitat en els sistemes, juntament amb la recerca per un estàndard per a fer tots aquests sistemes compatibles, ha suposat l'aparició de moltes propostes diferents.

Informació present en els protocol

Aquests protocols solen presentar la següent informació als *tickets* [7]:

- Número de serie.
- Emissor.
- Proveïdor del servei.
- Usuari.
- Servei.
- Termes de Condició.
- Tipus de tiquet.
- Destinació.
- Atribut.
- Temps de validesa.
- Data d'emissió.
- Signatura digital de l'emissor.
- Identificació del dispositiu.

Requeriments en els protocols

Els possibles requeriments d'un protocol d'aquestes característiques són [7]

- Integritat: Permet identificar si el contingut de l'*e-ticket* ha estat modificat.
- Autenticitat: L'usuari ha de poder verificar qui ha creat l'*e-ticket*.
- No rebuig en origen: Cap de les parts pot negar haver enviat un missatge que ha enviat.
- No rebuig en destí: Cap de les parts pot negar haver rebut un missatge que ha rebut.
- No falsificable: Només els usuaris autoritzats poden crear *e-tickets* vàlids.
- Just: Ninguna de les parts pot estar en una posició privilegiada.
- No *overspending*: Els *e-tickets* només es poden gastar les vegades acordades per les parts.
- Anonimat : Els *e-tickets* poden presentar diferents nivells d'anonimat:
  - Identificables: La identitat del propietari ha de ser verificable.
  - Anonimat completa revocable: L'anonimat dels usuaris pot ser revocat.
  - Anonimat selectiva revocable: La identitat d'un usuari fraudulent d'un usuari prèviament anònim pot ser coneguda.
  - Anònima total: L'usuari ha de sempre anònim.
- Exculpabilitat: El proveïdor de serveis no pot acusar un usuari honest d'*overspending*.
- Reutilitzable : L'*e-ticket* es pot emprar més d'una vegada.
- Transferible: Un usuari pot transferir *e-tickets* a altres usuaris.

2.2 Anàlisi de propostes de targetes ciutadanes

La majoria de la literatura estudiada es basa en la presentació d'una nova arquitectura, però molt poques de les propostes expliquen el protocol emprat per a dur-la a terme. A més a més, gairebé la majoria de les propostes estudiades no garanteixen l'anonimat de l'usuari ni la seguretat de les comunicacions.

A continuació s'explicaran una mostra de les diferents propostes estudiades.

2.2.1 MobiPag: Solució integrada de pagament mòbil, *ticketing* i cupons basats en NFC

En aquest article es proposa una plataforma amb l'objectiu "desmaterialitzar pagaments amb l'ús dels dispositius mòbils personals com un terminal de pagament personal que podrà ser interoperable entre diferents agents financers i proveïdors de comunicacions mòbils i capaç de ser adaptat per usuaris, comerciants, tendes i proveïdors de serveis"[8].

Aquest protocol està basat en el protocol EMV [9], desenvolupat per un conglomerat d'empreses (entre elles American Express, Visa i MasterCard) amb l'objectiu de crear un estàndard per als pagament segurs. Per a aquesta versió, el protocol s'ha simplificat eliminant la fase de negociació i afegint diverses característiques per a fer el protocol més versàtil, permetent no només la transacció de doblers sinó també la compra d'altres serveis com cupons, *tickets* i vals. A més a més, aquest sistema permet als comerciants afegir nous serveis, com *ticketing* o sistemes de fidelització.

El sistema requereix que l'usuari empri un dispositiu mòbil Android compatible amb tecnologia NFC i que contengui un dispositiu segur, mentre que el comerciant necessita un *Automated Payment Terminal* (ATP) i una aplicació *Point of Sale* (POS).

El principal desavantatge d'aquest sistema és el temps necessari per a validar els *tickets*, necessitant entre 15 i 30 segons, fent aquesta tecnologia difícil d'implementar per a l'entrada a una atracció o transport públic.

2.2.2 NFC *Smart Tourist Card*

NFC SMTC és un "assistent de viatge mòbil que integra les funcionalitats d'una *city card* tradicional amb les particularitats dels dispositius mòbils"[10].

Aquesta plataforma està orientada a turistes visitant diverses ciutats italianes. Aquesta tecnologia ofereix una Application Programming Interface (API) amb l'objectiu de crear un *framework* per a la creació de serveis compatibles entre ells. Per exemple, ofereix els serveis d'informació, pagament mòbil, *ticketing mòbil*, emparallament de dispositius, serveis segons localització, autorització d'accés i gestió de sistemes de fidelització i targetes d'afiliació.

Aquesta proposta està centrada en el potencial d'una aplicació d'aquestes característiques, però no explica ni l'arquitectura necessària per a dur-la a terme ni el protocol emprat per a realitzar les comunicacions entre els diferents actors.

2.2.3 Proposta d'una solució per a *ticketing* del transport públic basat en els dispositius mòbils dels clients

En aquest article es proposa un sistema de *ticketing* enfocat a les necessitats del proveïdor de transport públic de Porto, però amb la possibilitat d'adaptar-se a altres situacions [11].

Aquest sistema està basat en els "dispositius mòbils dels clients, que estan molt estesos i ofereixen moltes funcionalitats, i es basa en tecnologies de comunicacions mòbils, i proveïdors de localització com GPS i triangulació de xarxa"[11]. La plataforma permet al proveïdor de transport públic conèixer les costums de cada usuari, donant la possibilitat d'estudiar el comportaments dels usuaris i oferir ofertes personalitzades.

La particularitat d'aquest sistema és la manera com es comprova la validesa d'un *ticket*. L'usuari selecciona a quina aturada i línia vol redimir el seu *ticket* i pot viatjar per dins la zona seleccionada durant el temps de validesa del *ticket*. Per a validar si un usuari està viatjant amb un *ticket*, ha de mostrar la informació a la pantalla, un número de seqüència i un *watermarking* que canvia cada dia i el conductor coneix, amb l'objectiu d'evitar captures de pantalla.

El principal desavantatge d'aquesta plataforma és la seva limitació al sector del transport públic. A més a més, el mètode de facturació i la necessitat de comprovar explícitament la validesa de cada *ticket* suposa un nivell de confiança amb l'honradesa dels usuaris.

2.2.4 NFC4SURE: Sistema de *ticketing* mòbil

Tot i que la majoria de sistemes estudiats es centren en les funcionalitats que s'ofereixen a l'usuari, aquesta proposta es centra en la seguretat del sistema. La majoria de transaccions NFC necessiten de la presència de l'existència d'un element segur per guardar els credencials de l'usuari. Aquest sistema empra HCE (Host Card Emulation), en el qual el dispositiu es mòbil es connecta amb un element segur extern, i proposa un sistema amb les característiques següents [12]:

1. Securitzar els *tickets* mentre s'envien des de l'element segur extern a l'element encarregat de validar mitjançant l'*smartphone*.

2. El sistema no és vulnerable a *relay attacks*.

3. L'usuari es pot identificar davant l'element segur sense haver de confiar en l'*smartphone*.

El principal avantatge d'aquest sistema és la seva transparència, ja que l'arquitectura i operacions emprades són públiques, permetent a tercers comprovar els resultats obtinguts. Tot i així, la proposta es centra en la tecnologia NFC, limitant el seu possible ús a dispositius compatibles.

CAPÍTOL 3

PROTOCOL MCITYPASS

El protocol mCITYPASS és una proposta que presenta un sistema flexible que permet a l'usuari accedir de forma centralitzada en una única aplicació diferents serveis, reusables o no, amb el seu propi dispositiu (*smartphone*). A més a més, aquest protocol garanteix la seguretat i la privacitat de tot el sistema, així com l'anonimat dels usuaris honests. Aquest protocol queda definit per Payeras-Capellà et al. a l'article "*mCITYPASS : Privacy-preserving Secure Access to Federated Touristic Services with Mobile Devices*"[1].

Aquest protocol s'enfoca als *tickets* turístic electrònics, definint el procés per a comprar i emprar un fullet amb múltiples *tickets* per a accedir a atraccions turístiques i transport públic. Segons el serevi al qual permeten l'entrada, aquests *e-tickets* poden ser:

No reutilitzable: Només es pot accedir a aquest servei una vegada. Un servei no reutilitzable seria l'entrada a una atracció turística.

M **vegades reutilitzable:** Es pot accedir al servei un nombre de vegades definit. Un servei *m* vegades reutilitzable seria l'entrada al transport públic o una atracció turística.

Infinitament reutilitzable: Es pot accedir al servei un nombre infinit de vegades. Un servei infinitament reutilitzable seria l'entrada al transport públic.

Una de les característiques principals d'aquest protocol és la seva capacitat de suportar un sistema federat. Un sistema federat permet que els serveis disponibles siguin oferits per diferents proveïdors. Això permet als usuaris accedir a serveis molt diferents oferits per proveïdors públics o privats dins el mateix fullet. Tot i així, aquest protocol no contempla la repartició dels beneficis entre els diferents proveïdors.

Les solucions actuals basades en *smartcards* basen la seguretat del sistema en la seguretat física del *token*, però necessiten una inversió important per a crear la infraestructura per a la gestió de les *cards*. La implementació emprant els dispositius mòbils dels usuaris permet reduir la complexitat del sistema, alhora que simplifica

l'accés als usuaris. Aquest protocol ofereix els mateixos requeriments de seguretat per a una arquitectura basada en *smartphones* sense necessitar l'ús d'un dispositiu a prova de manipulacions.

3.1 Requeriments del protocol

A la Secció 2.1.3 s'han explicat les diferents opcions de seguretat que poden presentar un protocol d'aquestes característiques. D'aquests requeriments, aquest protocol complirà els següents:

- Autenticitat: L'usuari ha de poder verificar si un PASS ha estat creat per un *issuer* autoritzat.
- Integritat: Totes les parts han de poden verificar que un PASS no ha estat modificat des de que es va crear per l'*issuer*.
- No rebuig en origen: Una vegada creat un PASS, l'*issuer* no pot negar haver-l'ho creat.
- No falsificable: Només *issuers* autoritzats poden crear un PASS vàlid.
- No *overspending* i reutilitzable: Un PASS pot incloure diversos *tickets* amb diferents característiques:
  - No reutilitzable: L'usuari només pot emprar el servei una vegada.
  - M-vegades reutilitzable: L'usuari pot emprar el servei M vegades.
  - Infinitament reutilitzable: L'usuari pot emprar el servei totes les vegades que vulgui.

  En els dos primers casos el sistema ha de garantir que els *tickets* s'empren el nombre de vegades acordat.
- Anonimat revocable: L'usuari garanteix només l'anonimat dels usuaris que compleixen amb les condicions del servei. Si un usuari realitza una acció fraudulenta se'l pot identificar.
- Data d'expiració i temps de vida: El PASS només és vàlid abans de la data d'expiració i durant el temps de vida acordat.
- Activació: Abans de poder-se emprar, el PASS haurà de ser activat. Una vegada activat, començarà el temps de vida del PASS.
- Equitatiu: Ninguna de les parts pot estar en una posició privilegiada.
- Exculpabilitat: El *provider* no pot acusar a un usuari honest d'*overspending* i un usuari pot demostrar que ja ha validat el servei abans de fer ús del servei.
- No divisible: Només un usuari podrà validar *tickets* d'un determinat PASS.

Cal destacar que aquest protocol no permetrà transferir els PASS ja que tot i a ser anònims els PASS són personals. A més amés, segons Mut-Puigserver et. al. [7], aquesta característica és molt difícil d'assolir alhora que es garanteix el *no overspending*, especialment si també s'ha de garantir l'anonimat revocable dels usuaris.

3.2 Actors del protocol

3.2.1 *Trusted Third Party*

El *Trusted Third Party* és una entitat en la que confien tots els altres actors en un protocol encarregat de gestionar i resoldre possibles conflictes entre les diferents parts. En aquest protocol les funcions del Trusted Third Party (TTP) seran:

Donar d'alta als usuaris del sistema. El TTP és l'única entitat que coneixerà la identitat dels usuaris del sistema.

Garantir l'anonimat dels usuaris honests. El TTP generarà un pseudònim no vinculant per a garantir l'anonimat dels usuaris què vulguin fer ús del sistema.

Resoldre conflictes entre els diferents actors. En cas de què es produeixi un conflicte entre dues parts duran l'execució d'alguna de les fases del protocol, el TTP serà l'encarregat de gestionar aquestes situacions estudiant de manera imparcial les evidències que se li han presentat i deliberant a favor d'una de les dues parts. Les parts en conflicte hauran de resoldre la situació d'acord amb la decisió del TTP.

Revocar la identitat dels usuaris no honests. En cas de què un usuari faci un mal ús del sistema tan en l'execució del protocol com en l'ús del servei, el TTP podrà revocar la identitat de l'usuari.

3.2.2 *Issuer*

L' *Issuer* és l' entitat encarregada de gestionar els serveis disponibles dins el *framework* del mCITYPASS i emetre els PASS. Les seves funcions dins el protocol són:

Donar d'alta un servei. L'*Issuer* haurà de donar d'alta als *providers* abans de que aquests puguin oferir un servei.

Venta de PASS. L'*Issuer* és l'encarregat de crear tots els PASS vàlids per als usuaris. Els usuaris s'hauran de connectar amb ell i decidir la Categoria i el temps de vida del PASS. Només els PASS creats per l'*Issuer* seran vàlids.

Activació del PASS. Abans d'utilitzar un PASS, els *users* l'hauran d'activar davant l'*Issuer*. A partir d'aquest moment comença el període de vida útil del PASS.

3.2.3 *Provider*

Els *Providers* són els actors que ofereixen els serveis als usuaris. Abans de poder oferir aquest servei, l'han de donar d'alta contactant amb l'*Issuer*. Les seves funcions dins el protocol són:

Verificar els *tickets*. Els *providers* hauran de verificar que el *ticket* presentat per un *user* és vàlid i usable (no s'ha intentat emprar més vegades de les permeses) abans de poder accedir a un servei.

3.2.4 *User*

Els *Users* són els usuaris del *framework*. Abans de fer ús del sistema, els usuaris s'hauran de donar d'alta al TTP emprant la seva identitat real. Aquest garantirà el seu anonimat sempre que realitzin un ús correcte del sistema.

3.3 Fases del protocol

El protocol es divideix en cinc fases. Les dues primeres fases defineixen l'afiliació dels *providers* i a la generació dels pseudònims per als usuaris. Les tres darreres fan referència a la compra, activació i ús dels PASS.

3.3.1 Afiliació dels *providers*

Aquesta fase defineix l'afiliació dels *providers*, on aquests es donen d'alta amb l'*Issuer* i defineixen els serveis que oferiran dins el *framework* del mCITYPASS. Els *providers* poder oferir tres tipus de serveis:

- Servei no reutilitzable: El servei només es pot emprar una vegada.
- Servei `m`-vegades reutilitzable: El servei es pot emprar m vegades.
- Servei infinitament reutilitzable: El servei es pot emprar infinites vegades.

Les passes que seguiran l'*Issuer* i els *providers* i els missatges queden definits a la Figura 3.1.

Protocol: PROVIDERS' AFFILIATION

$\mathscr{I}$ chooses a certain public key algorithm (e.g., RSA, DSA) as a reference for the creation of P_i's public-key pairs.

authenticateProvider Provider P_i follows the next steps:

1. generates a public-key pair for a the chosen public key algorithm and the appropriate parameters;
2. sends this cryptographic information and the information of its service (either γ_i, λ_i or ξ_i depending on the reusability of the service) to $\mathscr{I}$ via an authenticated channel;

generateCertificate Issuer $\mathscr{I}$ executes:

1. verify that the information is correct;
2. generate a public key certificate for the P_i with the key usage (i.e., linked to the service γ_i or λ_i);

arrayGeneration Issuer $\mathscr{I}$ executes:

1. order and arrange the series of P_i's certificates in an array: $Cert_P[1..J]$;

Figura 3.1: Intercanvi d'informació entre l'*Issuer* i el *provider* [1].

Una vegada el *Provider* s'ha afiliat, es pot crear l'estructura d'un PASS. La Figura 3.2 mostra les llavors incloses en un PASS amb set serveis, un infinitament reutilitzable, tres serveis no reutilitzables i dos serveis quatre vegades reutilitzable.

Figura 3.2: Exemple d'un PASS

3.3.2 Generació del pseudònim dels *users*

Els *users* contacten amb el TTP per a generar un nou pseudònim abans de poder comprar un PASS. Aquest pseudònim garanteix l'anonimat de l'usuari alhora que evita la vinculabilitat en l'ús dels *tickets* anònims. Per a poder fer ús d'aquesta característica serà necessari que l'usuari torni a executar aquest protocol sempre que torni a la mateixa ciutat i torni a comprar un PASS.

Per a garantir l'anonimat dels usuaris, el sistema farà ús del protocol *Schnorr's Zero Knowledge Proof (ZKP)*. Aquest protocol suposa que les dues entitats coneixen els paràmetres públics que defineixen el protocol: un grup cíclic G d'ordre p amb generador α, on $p = 2q + 1$. Per a realitzar la comprovació es realitzen les operacions definides a la Figura 3.3.

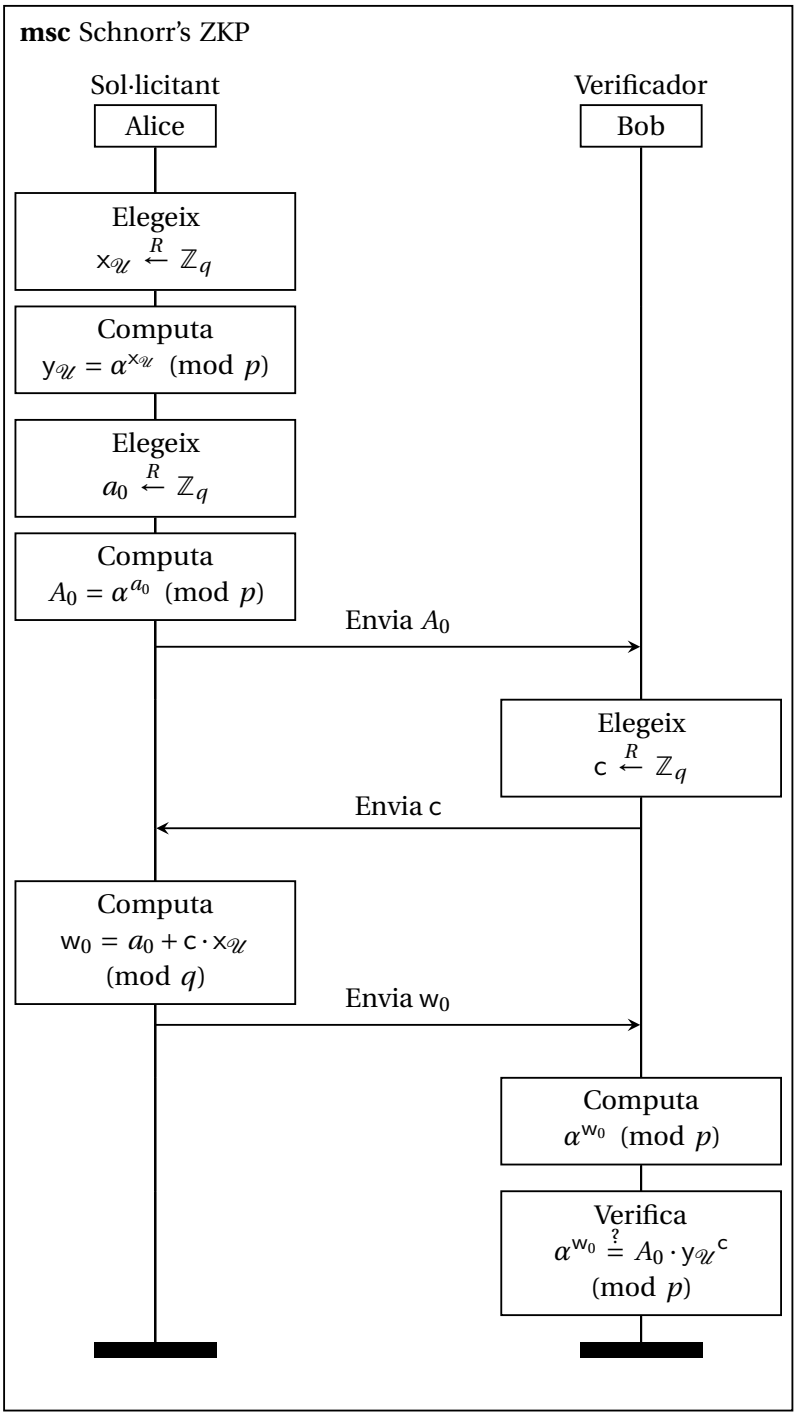

Figura 3.3: Seqüència de l'execució del protocol *Schnorr's ZKP*

Així doncs, el pseudònim d'un usuari quedarà definit pel valor $y_{\mathcal{U}}$ què calcularà durant l'intercanvi de missatges amb el TTP

Les passes que seguiran el TTP i els *users* i els missatges que s'intercanviaran queden definits a la Figura 3.4.

Protocol: PSEUDONYM

authenticateUser User $\mathcal{U}$ follows the next steps:

1. generates $x_{\mathcal{U}} \xleftarrow{R} \mathbb{Z}_q$, and computes $y_{\mathcal{U}} = \alpha^{x_{\mathcal{U}}} \pmod{p}$;
2. computes the signature $sk_{\mathcal{U}}(h_{y_{\mathcal{U}}})$ where $h_{y_{\mathcal{U}}} = hash(y_{\mathcal{U}})$;
3. encrypts $y_{\mathcal{U}}$ with the $\mathcal{T}$'s public key: $pk_{\mathcal{T}}(y_{\mathcal{U}})$;
4. sends $(sk_{\mathcal{U}}(h_{y_{\mathcal{U}}}), Cert_{\mathcal{U}}, pk_{\mathcal{T}}(y_{\mathcal{U}}))$ to $\mathcal{T}$;

generatePseudonym Pseudonym Manager $\mathcal{T}$ executes:

1. decrypts $sk_{\mathcal{T}}(pk_{\mathcal{T}}(y_{\mathcal{U}})) \rightarrow (y_{\mathcal{U}})$;
2. verifies $y_{\mathcal{U}}$: $pk_{\mathcal{U}}(sk_{\mathcal{U}}(h_{y_{\mathcal{U}}})) \rightarrow (h_{y_{\mathcal{U}}}) \stackrel{?}{=} hash(y_{\mathcal{U}})$;
3. if correct, then computes the signature of $sk_{\mathcal{T}}(h_{y_{\mathcal{U}}})$; and
4. sends $Pseu_{\mathcal{U}} = (y_{\mathcal{U}}, sk_{\mathcal{T}}(h_{y_{\mathcal{U}}}))$ to *user*.

verifyPseudonym $\mathcal{U}$ computes:

1. verifies $y_{\mathcal{U}}$: $pk_{\mathcal{T}}(sk_{\mathcal{T}}(h_{y_{\mathcal{U}}})) \rightarrow (h_{y_{\mathcal{U}}}) \stackrel{?}{=} hash(y_{\mathcal{U}})$;

Figura 3.4: Intercanvi d'informació entre el *user* i el TTP per generar el pseudònim [1].

3.3.3 Compra del PASS

El *user* es connecta amb l'*Issuer* i selecciona el tipus de PASS definint el *Lifetime* (temps de vida) i la Categoria (adult, jove, nin...). L'*Issuer* respon generant tota la informació necessària, així com un conjunt de valors aleatoris que els *providers* empraran per a atorgar a l'usuari el dret a emprar un servei.

És important destacar que per a garantir l'anonimat dels *users*, aquests no empren les seves claus personals sinó que s'autentiquen emprant els paràmetres del *Schnorr's ZKP* definits a la fase anterior.

Aquest protocol no considera el pagament del PASS.

Les passes que seguiran l'*Issuer* i els *users* i els missatges que s'intercanviaran queden definits a la Figura 3.5.

Protocol: PASS PURCHASE

getService $\mathscr{U}$ executes:

1. selects the desired Type of PASS Type(Lifetime, Category);
2. generates a of random value $\mathsf{RU} \xleftarrow{R} \mathbb{Z}_q$, and computes $\mathsf{h_{RU}} = hash(\mathsf{RU})$;
3. computes $\mathsf{H}_{\mathscr{U}} = \alpha^{\mathsf{RU}} \pmod{p}$;
4. generates two more random values $a_1, a_2 \xleftarrow{R} \mathbb{Z}_q$ to be used in the Schnorr proof;
5. computes $A_1 = \alpha^{a_1} \pmod{p}$;
6. computes $A_2 = \alpha^{a_2} \pmod{p}$;
7. sends $(\mathsf{Pseu}_{\mathscr{U}}, \mathsf{H}_{\mathscr{U}}, A_1, A_2, \mathsf{h_{RU}}, \mathsf{Type})$ to the ticket issuer $\mathscr{I}$.

getChallenge $\mathscr{I}$ follows the next steps:

1. generates and sends a challenge $\mathsf{c} \xleftarrow{R} \mathbb{Z}_q$ for $\mathscr{U}$;
2. asynchronously, for optimization, pre-computes $\mathsf{y}_{\mathscr{U}}{}^{\mathsf{c}} \pmod{p}$ and $\mathsf{H}_{\mathscr{U}}{}^{\mathsf{c}} \pmod{p}$;

solveChallenge $\mathscr{U}$ computes:

1. computes $\mathsf{w}_1 = a_1 + \mathsf{c} \cdot \mathsf{x}_{\mathscr{U}} \pmod{q}$;
2. computes $\mathsf{w}_2 = a_2 + \mathsf{c} \cdot \mathsf{RU} \pmod{q}$;
3. pre-computes the shared session key used in the ticket verification: $\mathsf{K} = hash(\mathsf{w}_2)$;
4. generates a set of J randoms, let J be the number of providers offering a service in the PASS, $\psi_{1,0}, ..., \psi_{J,0} \xleftarrow{R} \mathbb{Z}_q$
5. for each non-reusable service γ_i, calculates $\psi_{\gamma_i,1} = hash(\psi_{\gamma_i,0})$
6. for each reusable service λ_i (supose that m is the number of uses of service λ_i), computes $\psi_{\lambda_i,m} = hash^m(\psi_{\lambda_i,0})$ where $hash^{m+1}()$ represents the hash function applied m times
7. stores $\psi_{1,0}, ..., \psi_{J,0}$ values in her *TicketsPASS* database;
8. encrypts and sends the generated information to $\mathscr{I}$: $\mathsf{pk}_{\mathscr{I}}((\mathsf{w}_1, \mathsf{w}_2), (\{\psi_{\gamma_i,1}, \psi_{\lambda_i,m}\} \forall \gamma_i, \lambda_i))$ and pays for the PASS;

getPASS $\mathscr{I}$ follows the next steps:

1. decrypts $\mathsf{sk}_{\mathscr{I}}(\mathsf{pk}_{\mathscr{I}}(\mathsf{w}_1, \mathsf{w}_2)) \rightarrow (\mathsf{w}_1, \mathsf{w}_2)$;
2. computes $\alpha^{\mathsf{w}_1} \pmod{p}$;
3. computes $\alpha^{\mathsf{w}_2} \pmod{p}$;
4. verifies $\alpha^{\mathsf{w}_1} \stackrel{?}{=} A_1 \cdot \mathsf{y}_{\mathscr{U}}{}^{\mathsf{c}} \pmod{p}$;
5. verifies $\alpha^{\mathsf{w}_2} \stackrel{?}{=} A_2 \cdot \mathsf{H}_{\mathscr{U}}{}^{\mathsf{c}} \pmod{p}$;
6. if both verifications hold, $\mathscr{I}$ has checked that $\mathscr{U}$ is an authorized user. Otherwise, $\mathscr{I}$ stops the protocol;

7. computes the shared session key: $\mathsf{K} = hash(\mathsf{w}_2)$;
8. obtains a unique serial number Sn, and a set of random values $\mathsf{RI_i} \xleftarrow{R} \mathbb{Z}_p$, for $i = 0,,J$;
9. computes the set $\mathsf{h_{RI}}_i = hash(\mathsf{RI_i})$ for $i = 1,,J$;
10. composes the set $\kappa_i = (\mathsf{K}, \mathsf{RI_i})$ and signs it $\kappa_i^* = (\kappa_i, \mathsf{Sign}_{\mathscr{I}}(\kappa_i)) \forall i = 1,...,J$;
11. encrypts each κ_i^* with a digital envelope which is decryptable by the TTP $\mathscr{T}$ and the provider P_i for possible future controversial situations during the ticket verification: $\delta_{\mathscr{T},P_i} = \mathsf{pk}_{\mathscr{T},\mathscr{P}_i}(\kappa_i^*)$.
12. fills out the PASS information PASS = (Sn, Type, $\mathsf{Pseu}_{\mathscr{U}}$, Lifetime, PURdate, EXPdate, $\mathsf{h_{RI}}[\,]$, $\mathsf{h_{RU}}$, $H_{\mathscr{U}}$ $\delta_{\mathscr{T},P_i}[\,]$, $\psi_{\lambda,m}[\,]$, $\psi_{\gamma,1}[\,]$, Terms and Conditions);
13. digitally signs the PASS, $\mathsf{Sign}_{\mathscr{I}}(\mathsf{PASS}) = \mathsf{sk}_{\mathscr{I}}(hash(\mathsf{PASS}))$, and generates $\mathsf{PASS}^* = (\mathsf{PASS}, \mathsf{Sign}_{\mathscr{I}}(\mathsf{PASS}))$;
14. stores in its *CityPASS* database the information related to this ticket: $(\mathsf{PASS}^*, \kappa_i \quad \forall i = 1,...,J)$
15. sends PASS^* to the user $\mathscr{U}$.

receivePASS $\mathscr{U}$ executes:

1. verifies the digital signature $\mathsf{Sign}_{\mathscr{I}}(\mathsf{PASS})$ of the PASS using the issuer's certificate;
2. verifies PASS data and if the performed request match;
3. verifies the PASS validity $(\mathsf{PASS.PURdate}, \mathsf{PASS.EXPdate})$;
4. verifies $\mathsf{PASS.Pseu}_{\mathscr{U}}$;
5. stores in her *TicketsPASS* database $(\mathsf{PASS}^*, \mathsf{RU})$ together with the associated information stored in the step **solveChallenge**.7 of this phase.

Figura 3.5: Intercanvi d'informació entre el *user* i l'*Issuer* per a la compra del PASS [1].

La generació de les llavors a partir dels paràmetres de les arrels es pot observar a la Figura 3.6. Aquesta figura mostra la generació d'aquests paràmetres inclosos en, aquest cas, un PASS amb set serveis, un infinitament reutilitzable, tres serveis no reutilitzables i dos serveis quatre vegades reutilitzable.

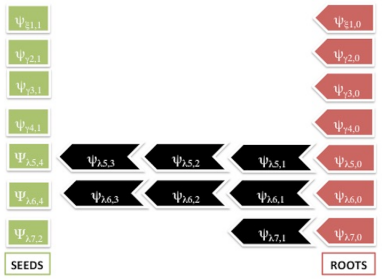

Figura 3.6: Generació d'un PASS

3.3.4 Activació del PASS

Els PASS es podran emprar en qualsevol moment abans de la seva data d'expiració i durant el seu *Lifetime*. Aquest *Lifetime* comença en el moment quan l'usuari activa el PASS. Aquest procés es pot dur a terme en qualsevol moment entre la compra del PASS i el primer ús d'aquest.

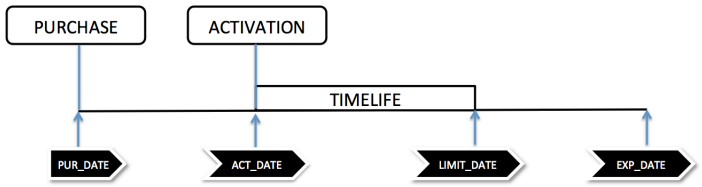

Figura 3.7: Cicle de vida d'un PASS

Com es pot observar a la Figura 3.7, cada PASS queda definit temporalment per cinc paràmetres:

Timelife: Període de temps durant el qual es pot emprar el PASS des de l'activació.

Purchase Date: Data de compra del PASS.

Activation Date: Data d'activació del PASS.

Limit Date: Data límit per a emprar el PASS. Aquesta data es calcula sumant el *Lifetime* a la data d'activació.

Expiration Date: Data límit per a emprar els serveis del PASS. Si la **Limit Date** és després d'aquesta data, els serveis només es podran emprar fins l'**Expiration Date**.

Les passes que seguiran l'*Issuer* i els *users* i els missatges que s'intercanviaran queden definits a la Figura 3.8.

Protocol 1: PASS ACTIVATION

showPass $\mathscr{U}$ computes:

1. sends PASS* to $\mathscr{I}$;

verifyTicket $\mathscr{I}$ executes:

1. verifies the PASS signature, PASS.PURdate, and PASS.EXPdate;
2. if the verifications fail, $\mathscr{I}$ aborts the ticket activation;
3. else $\mathscr{I}$ looks for the PASS PASS* in the database using PASS.Sn; and verifies that the PASS has not been activated: $\stackrel{?}{\nexists}$ ACT linked to PASS* in the DB;
   a) if $\nexists$ACT linked to PASS*:

i. generates ACT = (PASS.Sn, ACTdate, "$Activated$") and digitally signs ACT, and obtains the signed activation proof, $\text{Sign}_{\mathscr{I}}(\text{ACT}) = \text{sk}_{\mathscr{I}}(hash(\text{ACT}))$, and $\text{ACT}^* = (\text{ACT}, \text{Sign}_{\mathscr{I}}(\text{ACT}))$;
ii. sends ACT^* to the user $\mathscr{U}$.

Figura 3.8: Intercanvi d'informació entre l'*Issuer* i el *user* per a activar un PASS [1].

3.3.5 Verificació del PASS

L'usuari haurà de demostrar que és el propietari d'un PASS vàlid i activat. Durant aquesta fase, el *user* només interactua amb el *provider* que ofereix el servei al qual hi vol accedir. Només en cas de conflicte, el *user* o el *Provider* podran contactar amb el TTP emprant un canal resistent (el missatge sempre arribarà) per a poder garantir els requeriments de seguretat del protocol. En cas de que l'*user* no faci un ús correcte del servei, es pot revocar el seu anonimat.

Per a poder verificar un PASS, el *user* haurà de demostrar que coneix el valor RU associat al PASS emprant ZKP i, a canvi, obtendrà el valor RI, el qual li donarà accés al servei.

El procés a seguir canviarà segons el tipus de servei al qual vol accedir el *user* (no re-usable, m vegades re-usable o infinitament re-usable).

La verificació del *ticket* per a accedir a un servei no re-usable queda definit a la Figura 3.9.

Protocol: Verification of a non-reusable service

The service *provider* P_i and *user* $\mathscr{U}$ follow these steps:

showPASS $\mathscr{U}$ computes:

1. generates a random value $a_3, \overset{R}{\leftarrow} \mathbb{Z}_q$ to be used in a Schnorr proof;
2. computes $A_3 = \alpha^{a_3} \pmod{p}$;
3. compose the information ticket message $\text{m}_1 = (\text{PASS}^*, \text{ACT}^*, A_3)$;
4. signs and sends it to *providers*: $\text{m}_1{}^* = (\text{m}_1, \text{sk}_{\mathscr{P}i}(hash(\text{m}_1)))$;

verifyPASS P_i executes:

1. verifies the PASS signature, PASS.Sv, PASS.PURdate, and PASS.EXPdate;
2. verifies ACT^*, and PASS.ACTdate
3. calculates LIMdate = ACTdate + Lifetime
4. verifies that the present date is not greater that LIMdate

→ **if any verification fails:**

5. assigns $\text{V}_{\text{fail}} = (\text{PASS.Sn}, \text{flag}_{00}, \tau_1)$.
6. signs $\text{V}_{\text{fail}}{}^* = (\text{V}_{\text{fail}}, \text{sk}_{\mathscr{P}i}(hash(\text{V}_{\text{fail}})))$ and sends $\text{m}_2 = \text{V}_{\text{fail}}{}^*$ to $\mathscr{U}$

→ **else:**

5. looks for the PASS PASS^* in its *SpentCityPASSES* database using $\mathsf{PASS.Sn}$ and verifies that the ticket has not been spent:
   a) if $\nexists \psi_{\gamma_i,0}$ linked to PASS^* in the database:
   P_i follows the next steps:
      i. generates a challenge $\mathsf{c} \xleftarrow{R} \mathbb{Z}_q$;
      ii. assigns $Challenge = (\mathsf{PASS.Sn}, \mathsf{c}, \tau_1)$;
      iii. $Challenge^* = (Challenge, \mathsf{sk}_{\mathscr{P}i}(hash(Challenge)))$ and sends this signature to $\mathscr{U}$;
      iv. asynchronously, for optimization, pre-computes $\mathsf{H}_{\mathscr{U}}{}^{\mathsf{c}} \pmod p$;

   $\mathscr{U}$ computes:
      i. computes $\mathsf{w}_3 = a_3 + \mathsf{c} \cdot \mathsf{RU} \pmod q$;
      ii. encrypts and signs w_3 and, then, sends it to P_i: $\mathsf{sk}_{\mathscr{U}}(\mathsf{pk}_{\mathscr{P}i}(\mathsf{PASS.Sn}, \mathsf{w}_3, \tau_1))$;

   P_i follows the next steps:
      i. computes $\alpha^{\mathsf{w}_3} \pmod p$;
      ii. verifies $\alpha^{\mathsf{w}_3} \stackrel{?}{=} A_3 \cdot \mathsf{H}_{\mathscr{U}}{}^{\mathsf{c}} \pmod p$;
      → **if verification fails:**
      iii. assigns $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{01}, \tau_1)$.
      iv. signs $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{fail}})))$ and sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathscr{U}$
      → **else:**
      iii. computes $\mathsf{A}_{\mathscr{P}} = PRNG(\mathsf{h_K}) \oplus \mathsf{RI}_{\gamma_i}$, where $PRNG(\mathsf{h_K})$ is a secure pseudorandom number generator and, $\mathsf{h_K} = hash(\mathsf{K})$ is the seed. Note that K and RI_{γ_i} are obtained from $\delta_{\mathscr{T},P_i}$;
      iv. encrypts $\mathsf{A}_{\mathscr{P}}$ with the public key of the TTP $\mathscr{T}$: $\mathsf{pk}_{\mathscr{T}}(\mathsf{A}_{\mathscr{P}})$;
      v. assigns $\mathsf{V_{succ}} = (\gamma_i, \mathsf{PASS.Sn}, \mathsf{flag}_{10}, \tau_1, \mathsf{pk}_{\mathscr{T}}(\mathsf{A}_{\mathscr{P}}))$, ($\tau_1$ is the verification timestamp). The signature is noted: $\mathsf{V_{succ}}^* = (\mathsf{V_{succ}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{succ}})))$;
      vi. sends $\mathsf{m}_2 = \mathsf{V_{succ}}^*$ to *user*;
   b) if $\exists \psi_{\gamma_i,0}$ linked to PASS^* in the database:
      i. assigns $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \psi_{\gamma_i,0}, \mathsf{flag}_{02}, \tau_1, \gamma_i)$.The signature is noted:
      $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{fail}})))$;
      ii. sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathscr{U}$; indicating that the this non reusable ticket of the PASS had been already used.

showProof $\mathscr{U}$ executes:

1. verifies P_i's signature;
→ **if** $\mathsf{V_{fail}}^*$ **is received or** $\mathsf{V_{succ}}^*$ **is not correct:**
2. *user* checks the appropriate flag to know the details of the error. If he does not agree then he can rise a CLAIM by contacting with $\mathscr{T}$;
→ **else:**

2. calculates $\mathsf{A}_{\mathscr{U}} = PRNG(\mathsf{K}) \oplus \psi_{\gamma_i,0}$, using the shared value K as seed;
3. compose the message $\mathsf{m}_3 = (\mathsf{PASS.Sn}, \mathsf{A}_{\mathscr{U}})$;
4. signs and sends it to P_i: ${\mathsf{m}_3}^* = (\mathsf{m}_3, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{m}_3)))$;

verifyProof P_i follows the next steps:

1. obtains PASS.Sn, and computes $\psi_{\gamma_i,0} = \mathsf{A}_{\mathscr{U}} \oplus PRNG(\mathsf{K})$;
2. verifies $\psi_{\gamma_i,1} \stackrel{?}{=} hash(\psi_{\gamma_i,0})$;
3. generates τ_2 and verifies it using the PASS expiry date (PASS.PURdate, PASS.EXPdate) and the timestamp τ_1, the value of ACTdate of the element ACT and LIMdate, being LIMdate = ACTdate + Lifetime;

→ **if any verification fails:**

4. assigns $\mathsf{V}_{\mathsf{fail}} = (\gamma_i, \mathsf{PASS.Sn}, \mathsf{flag}_{03}, \tau_2, \psi_{\gamma_i,0})$.
5. signs ${\mathsf{V}_{\mathsf{fail}}}^* = (\mathsf{V}_{\mathsf{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V}_{\mathsf{fail}})))$, sends $\mathsf{m}_2 = {\mathsf{V}_{\mathsf{fail}}}^*$ to $\mathscr{U}$

→ **else:**

4. signs $\mathsf{A}_{\mathscr{P}}$ approving then the validation with timestamp τ_2: $\mathsf{R}_{\gamma_i} = (\mathsf{A}_{\mathscr{P}}, \mathsf{PASS.Sn}, \tau_2)$, and ${\mathsf{R}^*}_{\gamma_i} = (\mathsf{R}_{\gamma_i}, \mathsf{sk}_{\mathscr{P}}(hash(\mathsf{R}_{\gamma_i})))$;
5. stores in its *SpentCityPASSES* database: $(\mathsf{PASS}^*, \psi_{\gamma_i,0})$ and sends $\mathsf{m}_4 = {\mathsf{R}^*}_{\gamma_i}$ to $\mathscr{U}$;

getValidationConfirmation $\mathscr{U}$ follows the next steps:

→ **if** ${\mathsf{V}_{\mathsf{fail}}}^*$ **is received:**

1. $\mathscr{U}$ checks flag_{03} to know the details of the error. If he does not agree then he can rise a CLAIM by contacting with the $\mathscr{T}$;

→ **else:**

1. checks the signature of ${\mathsf{R}^*}_{\gamma_i}$;
2. computes $\mathsf{RI}_{\gamma_i} = \mathsf{A}_{\mathscr{P}} \oplus PRNG(\mathsf{h}_{\mathsf{K}})$;
3. verifies $\mathsf{h}_{\mathsf{RI}_{\gamma_i}} \stackrel{?}{=} hash(\mathsf{RI}_{\gamma_i})$;

→ **if any verification fails:**

4. $\mathscr{U}$ collects all evidence and he can rise a CLAIM by contacting with the $\mathscr{T}$, he can argue that there is an error in getting the authorisation to use the service (*Authorisation Failure*);

→ **else:**

4. stores in her *TicketsPASS* database $({\mathsf{R}^*}_{\gamma_i}, \mathsf{RI}_{\gamma_i})$ together with PASS^*.

Figura 3.9: Intercanvi d'informació entre el *provider* i el *user* per a verificar un servei no re-usable [1].

La verificació del *ticket* per a accedir a un servei m vegades re-usable queda definit a la Figura 3.10.

Protocol: Verification of a m-times reusable service

The service *provider* P_i and *user* $\mathscr{U}$ follow these steps:

showPASS $\mathscr{U}$ computes:

1. generates a random value $a_3, \xleftarrow{R} \mathbb{Z}_q$ to be used in a Schnorr proof;
2. computes $A_3 = \alpha^{a_3} \pmod{p}$;
3. **Case 1-First time use of a m-times reusable service:** *user* computes $\psi_{\lambda_i,m-1} = hash^{m-1}(\psi_{\lambda_i,0})$. Then, $\mathscr{U}$ creates a counter $k_{\lambda_i} = m-1$ to keep track of the number of times that the ticket can be used. The counter is stored in her *CityPASS* database together with the rest of the information associated to this PASS ticket[a].
3. **Case 2-Subsequent use of a m-times reusable service:** *user* retrieves k_{λ_i} from her *TicketsPASS* database and computes $\psi_{\lambda_i,(k_{\lambda_i}-1)} = hash^{(k_{\lambda_i}-1)}(\psi_{\lambda_i,0})$.
4. compose the information ticket message $\mathsf{m}_1 = (\mathsf{PASS}^*, \mathsf{ACT}^*, A_3, (k_{\lambda_i}))$;
5. signs and sends it to P_i: ${\mathsf{m}_1}^* = (\mathsf{m}_1, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{m}_1)))$;

verifyPASS P_i executes:

1. verifies the PASS signature, PASS.PURdate, and PASS.EXPdate;
2. verifies ACT^*, and PASS.ACTdate;

→ **if any verification fails:**

3. assigns $\mathsf{V}_{\mathsf{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{00}, \tau_1)$.
4. signs ${\mathsf{V}_{\mathsf{fail}}}^* = (\mathsf{V}_{\mathsf{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V}_{\mathsf{fail}})))$ and sends $\mathsf{m}_2 = {\mathsf{V}_{\mathsf{fail}}}^*$ to $\mathscr{U}$

→ **else:**

3. P_i looks for the PASS^* in its *SpentCityPASSES* database using PASS.Sn; and verifies that the ticket has not been already spent m-times:
   a) if ($\nexists \psi_{\lambda,k_{\lambda_i}}$ linked to PASS^*) or [($\exists \psi_{\lambda,k_{\lambda_i}}$) and ($k_{\lambda_i} \geq 1$) and ($k_{\lambda_i}$ stored in the P_i database has the same value than the one sent by $\mathscr{U}$)]:
   P_i follows the next steps:
      i. generates a challenge $\mathsf{c} \xleftarrow{R} \mathbb{Z}_q$;
      ii. assigns $Challenge = (\mathsf{PASS.Sn}, \mathsf{c}, \tau_1)$;
      iii. ${Challenge}^* = (Challenge, \mathsf{sk}_{\mathscr{P}i}(hash(Challenge)))$ and sends this signature to $\mathscr{U}$;
      iv. asynchronously, for optimization, pre-computes ${\mathsf{H}_{\mathscr{U}}}^{\mathsf{c}} \pmod{p}$;

   $\mathscr{U}$ computes:
      i. computes $\mathsf{w}_3 = a_3 + \mathsf{c} \cdot \mathsf{RU} \pmod{q}$;
      ii. encrypts and signs w_3 and, then, sends it to P_i: $\mathsf{sk}_{\mathscr{U}}(\mathsf{pk}_{\mathscr{P}i}(\mathsf{PASS.Sn}, \mathsf{w}_3, \tau_1))$;

   P_i follows the next steps:
      i. computes $\alpha^{\mathsf{w}_3} \pmod{p}$;
      ii. verifies $\alpha^{\mathsf{w}_3} \stackrel{?}{=} A_3 \cdot {\mathsf{H}_{\mathscr{U}}}^{\mathsf{c}} \pmod{p}$;

   → **if verification fails:**

iii. assigns $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{01}, \tau_1)$.
iv. signs $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{fail}})))$ and sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to *user*

→ **else:**

iii. computes $\mathsf{A}_{\mathscr{P}} = PRNG(\mathsf{h_K}) \oplus \mathsf{RI}_{\lambda_i}$, where $PRNG(\mathsf{h_K})$ is a secure pseudorandom number generator and, $\mathsf{h_K} = hash(\mathsf{K})$ is the seed. Note that K and RI_{λ_i} are obtained from $\delta_{\mathscr{T},P_i}$;
iv. encrypts $\mathsf{A}_{\mathscr{P}}$ with the public key of the TTP $\mathscr{T}$: $\mathsf{pk}_{\mathscr{T}}(\mathsf{A}_{\mathscr{P}})$;
v. assigns $\mathsf{V_{succ}} = (\lambda_i, \mathsf{PASS.Sn}, \mathsf{flag}_{10}, \tau_1, \mathsf{pk}_{\mathscr{T}}(\mathsf{A}_{\mathscr{P}}), k_{\lambda_i}, \psi_{(\lambda,k_{\lambda_i})})$. The signature is noted: $\mathsf{V_{succ}}^* = (\mathsf{V_{succ}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{succ}})))$;
vi. sends $\mathsf{m}_2 = \mathsf{V_{succ}}^*$ to *user*;

b) else[b]:

i. assigns $\mathsf{V_{fail}} = (\lambda_i, \mathsf{PASS.Sn}, \mathsf{flag}_{04}, \tau_1, k_{\lambda_i}, \psi_{(\lambda,k_{\lambda_i})})$. The signature is noted: $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathscr{P}}(hash(\mathsf{V_{fail}})))$;
ii. sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathscr{U}$;

showProof $\mathscr{U}$ executes:

1. verifies P_i's signature;

→ **if $\mathsf{V_{fail}}^*$ is received or $\mathsf{V_{succ}}^*$ is not correct:**

2. $\mathscr{U}$ checks the appropriate flag to know the details the error. If he does not agree then he can rise a CLAIM by contacting with $\mathscr{T}$;

→ **else:**

2. calculates $\mathsf{A}_{\mathscr{U}} = PRNG(\mathsf{K}) \oplus \psi_{\lambda,(k_{\lambda_i}-1)}$, using the shared value K as seed;
3. compose the message $\mathsf{m}_3 = (\mathsf{PASS.Sn}, \mathsf{A}_{\mathscr{U}})$;
4. signs and sends it to *providers*: $\mathsf{m_3}^* = (\mathsf{m}_3, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{m}_3)))$;

verifyProof P_i follows the next steps:

1. obtains PASS.Sn, and computes $\psi_{\lambda,(k_{\lambda_i}-1)} = \mathsf{A}_{\mathscr{U}} \oplus PRNG(\mathsf{K})$;
2. checks $\psi_{\lambda,(k_{\lambda_i})} \stackrel{?}{=} hash(\psi_{\lambda,(k_{\lambda_i}-1)})$,
3. generates τ_2 and verifies it using the PASS expiry date (PASS.PURdate, PASS.EXPdate) and the timestamp τ_1;

→ **if any verification fails:**

4. assigns $\mathsf{V_{fail}} = (\lambda_i, \mathsf{PASS.Sn}, \mathsf{flag}_{05}, \tau_2, \psi_{\lambda,(k_{\lambda_i}-1)}, (k_{\lambda_i}-1))$.
5. signs $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{fail}})))$, sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathscr{U}$

→ **else:**

4. signs $\mathsf{A}_{\mathscr{P}}$ approving then the validation with timestamp τ_2: $\mathsf{R}_{\lambda_i} = (\mathsf{A}_{\mathscr{P}}, \mathsf{PASS.Sn}, \tau_2)$, and $\mathsf{R^*}_{\lambda_i} = (\mathsf{R}_{\lambda_i}, \mathsf{sk}_{\mathscr{P}}(hash(\mathsf{R}_{\lambda_i})))$;
5. stores in the *SpentCityPASSES* database: $(\mathsf{PASS}^*, \psi_{\lambda,(k_{\lambda_i}-1)})$;
6. updates the value of $k_{\lambda_i} = (k_{\lambda_i}-1)$ and stored it with the ticket information;
7. sends $\mathsf{m}_4 = \mathsf{R^*}_{\lambda_i}$ to $\mathscr{U}$;

getValidationConfirmation $\mathscr{U}$ follows the next steps:

→ **if** $\mathsf{V_{fail}}^*$ **is received:**

1. $\mathscr{U}$ checks flag to know the details the error. If he does not agree then he can rise a CLAIM by contacting with the $\mathscr{T}$;

→ **else:**

1. checks the signature of $\mathsf{R^*}_{\lambda_i}$;
2. computes $\mathsf{RI}_{\lambda_i} = \mathsf{A}_{\mathscr{P}} \oplus PRNG(\mathsf{h_K})$;
3. verifies $\mathsf{h}_{\mathsf{RI}_{\lambda_i}} \stackrel{?}{=} hash(\mathsf{RI}_{\lambda_i})$;

→ **if any verification fails:**

4. $\mathscr{U}$ collects evidences and he can rise a CLAIM contacting with $\mathscr{T}$;

→ **else:**

4. stores in her *TicketsPASS* database $(\mathsf{R^*}_{\lambda_i}, \mathsf{RI}_{\lambda_i})$ together with PASS^* and $k_{\lambda_i} = (k_{\lambda_i} - 1)$.

[a] Note that $\psi_{\lambda_i,0}$ was stored by *user* in her *TicketsPASS* database at the PASS purchase phase.
[b] An error has been detected, and thus the ticket is not valid. The reason of the error can be:

- k_{λ_i} sent by *user* is greater than the one stored by *providers*
- $\psi_{(\lambda,k_{\lambda_i}-1)}$ is not correct
- the *PASS* for the λ_i service is over $(k_{\lambda_i} = 0)$

Figura 3.10: Intercanvi d'informació entre el *provider* i el *user* per a verificar un servei *m* vegades re-usable [1].

La verificació del *ticket* per a accedir a un servei infinitament re-usable és molt similar al procediment seguit per a comprovar un *ticket* per a un servei no re-usable. L'única diferència es produeix quan el *provider* verifica el PASS, quan només comprova la validesa de les dates i no consulta si aquest servei ja ha estat emprat.

El procediment exacte emprat per a la primera vegada que s'empra el servei queda definit a la Figura 3.9.

Protocol: Verification of an infinite times reusable service

The service *provider* P_i and *user* $\mathscr{U}$ follow these steps:

showPASS $\mathscr{U}$ computes:

1. generates a random value $a_3, \stackrel{R}{\leftarrow} \mathbb{Z}_q$ to be used in a Schnorr proof;
2. computes $A_3 = \alpha^{a_3} \pmod{p}$;
3. compose the information ticket message $\mathsf{m}_1 = (\mathsf{PASS}^*, \mathsf{ACT}^*, A_3)$;
4. signs and sends it to *providers*: ${\mathsf{m}_1}^* = (\mathsf{m}_1, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{m}_1)))$;

verifyPASS P_i executes:

1. verifies the PASS signature, PASS.Sv, PASS.PURdate, and PASS.EXPdate;

2. verifies ACT^*, and PASS.ACTdate
3. calculates LIMdate = ACTdate + Lifetime
4. verifies that the present date is not greater that LIMdate

→ **if any verification fails:**

5. assigns $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{00}, \tau_1)$.
6. signs ${\mathsf{V_{fail}}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{fail}})))$ and sends $\mathsf{m}_2 = {\mathsf{V_{fail}}}^*$ to $\mathscr{U}$

→ **else:**

5. generates a challenge $\mathsf{c} \xleftarrow{R} \mathbb{Z}_q$;
6. assigns $Challenge = (\mathsf{PASS.Sn}, \mathsf{c}, \tau_1)$;
7. $Challenge^* = (Challenge, \mathsf{sk}_{\mathscr{P}i}(hash(Challenge)))$ and sends this signature to $\mathscr{U}$;
8. asynchronously, for optimization, pre-computes ${\mathsf{H}_{\mathscr{U}}}^{\mathsf{c}} \pmod p$; $\mathscr{U}$ computes:
   a) computes $\mathsf{w}_3 = a_3 + \mathsf{c} \cdot \mathsf{RU} \pmod q$;
   b) encrypts and signs w_3 and, then, sends it to P_i: $\mathsf{sk}_{\mathscr{U}}(\mathsf{pk}_{\mathscr{P}i}(\mathsf{PASS.Sn}, \mathsf{w}_3, \tau_1))$;

   P_i follows the next steps:
   a) computes $\alpha^{\mathsf{w}_3} \pmod p$;
   b) verifies $\alpha^{\mathsf{w}_3} \stackrel{?}{=} A_3 \cdot {\mathsf{H}_{\mathscr{U}}}^{\mathsf{c}} \pmod p$;

   → **if verification fails:**

   iii. assigns $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{01}, \tau_1)$.
   iv. signs ${\mathsf{V_{fail}}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{fail}})))$ and sends $\mathsf{m}_2 = {\mathsf{V_{fail}}}^*$ to $\mathscr{U}$

   → **else:**

   c) computes $\mathsf{A}_{\mathscr{P}} = PRNG(\mathsf{h_K}) \oplus \mathsf{RI}_{\xi_i}$, where $PRNG(\mathsf{h_K})$ is a secure pseudorandom number generator and, $\mathsf{h_K} = hash(\mathsf{K})$ is the seed. Note that K and RI_{ξ_i} are obtained from $\delta_{\mathscr{T},P_i}$;
   d) encrypts $\mathsf{A}_{\mathscr{P}}$ with the public key of the TTP $\mathscr{T}$: $\mathsf{pk}_{\mathscr{T}}(\mathsf{A}_{\mathscr{P}})$;
   e) assigns $\mathsf{V_{succ}} = (\xi_i, \mathsf{PASS.Sn}, \mathsf{flag}_{10}, \tau_1, \mathsf{pk}_{\mathscr{T}}(\mathsf{A}_{\mathscr{P}}))$, ($\tau_1$ is the verification timestamp). The signature is noted: ${\mathsf{V_{succ}}}^* = (\mathsf{V_{succ}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{succ}})))$;
   f) sends $\mathsf{m}_2 = {\mathsf{V_{succ}}}^*$ to *user*;
9. if $\exists \psi_{\xi_i,0}$ linked to PASS^* in the database:
   a) assigns $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \psi_{\xi_i,0}, \mathsf{flag}_{02}, \tau_1, \xi_i)$.The signature is noted: ${\mathsf{V_{fail}}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V_{fail}})))$;
   b) sends $\mathsf{m}_2 = {\mathsf{V_{fail}}}^*$ to $\mathscr{U}$; indicating that the this non reusable ticket of the PASS had been already used.

showProof $\mathscr{U}$ executes:

1. verifies P_i's signature;

→ **if** ${\mathsf{V_{fail}}}^*$ **is received or** ${\mathsf{V_{succ}}}^*$ **is not correct:**

2. *user* checks the appropriate flag to know the details of the error. If he does not agree then he can rise a CLAIM by contacting with $\mathscr{T}$;

→ **else:**

2. calculates $\mathsf{A}_{\mathscr{U}} = PRNG(\mathsf{K}) \oplus \psi_{\xi_i,0}$, using the shared value K as seed;
3. compose the message $\mathsf{m}_3 = (\mathsf{PASS.Sn}, \mathsf{A}_{\mathscr{U}})$;
4. signs and sends it to P_i: ${\mathsf{m}_3}^* = (\mathsf{m}_3, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{m}_3)))$;

verifyProof P_i follows the next steps:

1. obtains PASS.Sn, and computes $\psi_{\xi_i,0} = \mathsf{A}_{\mathscr{U}} \oplus PRNG(\mathsf{K})$;
2. verifies $\psi_{\xi_i,1} \stackrel{?}{=} hash(\psi_{\xi_i,0})$;
3. generates τ_2 and verifies it using the PASS expiry date (PASS.PURdate, PASS.EXPdate) and the timestamp τ_1, the value of ACTdate of the element ACT and LIMdate, being LIMdate = ACTdate + Lifetime;

→ **if any verification fails:**

4. assigns $\mathsf{V}_{\mathsf{fail}} = (\xi_i, \mathsf{PASS.Sn}, \mathsf{flag}_{03}, \tau_2, \psi_{\xi_i,0})$.
5. signs ${\mathsf{V}_{\mathsf{fail}}}^* = (\mathsf{V}_{\mathsf{fail}}, \mathsf{sk}_{\mathscr{P}i}(hash(\mathsf{V}_{\mathsf{fail}})))$, sends $\mathsf{m}_2 = {\mathsf{V}_{\mathsf{fail}}}^*$ to $\mathscr{U}$

→ **else:**

4. signs $\mathsf{A}_{\mathscr{P}}$ approving then the validation with timestamp τ_2: $\mathsf{R}_{\xi_i} = (\mathsf{A}_{\mathscr{P}}, \mathsf{PASS.Sn}, \tau_2)$, and ${\mathsf{R}^*}_{\xi_i} = (\mathsf{R}_{\xi_i}, \mathsf{sk}_{\mathscr{P}}(hash(\mathsf{R}_{\xi_i})))$;
5. stores in its *SpentCityPASSES* database: $(\mathsf{PASS}^*, \psi_{\xi_i,0})$ and sends $\mathsf{m}_4 = {\mathsf{R}^*}_{\xi_i}$ to $\mathscr{U}$;

getValidationConfirmation $\mathscr{U}$ follows the next steps:

→ **if** ${\mathsf{V}_{\mathsf{fail}}}^*$ **is received:**

1. $\mathscr{U}$ checks flag_{03} to know the details of the error. If he does not agree then he can rise a CLAIM by contacting with the $\mathscr{T}$;

→ **else:**

1. checks the signature of ${\mathsf{R}^*}_{\xi_i}$;
2. computes $\mathsf{RI}_{\xi_i} = \mathsf{A}_{\mathscr{P}} \oplus PRNG(\mathsf{h}_{\mathsf{K}})$;
3. verifies $\mathsf{h}_{\mathsf{RI}_{\xi_i}} \stackrel{?}{=} hash(\mathsf{RI}_{\xi_i})$;

→ **if any verification fails:**

4. $\mathscr{U}$ collects all evidence and he can rise a CLAIM by contacting with the $\mathscr{T}$, he can argue that there is an error in getting the authorisation to use the service (*Authorisation Failure*);

→ **else:**

4. stores in her *TicketsPASS* database $({\mathsf{R}^*}_{\xi_i}, \mathsf{RI}_{\xi_i})$ together with PASS^*.

Figura 3.11: Intercanvi d'informació entre el *provider* i el *user* per a verificar un servei infinitament reutilitzable [1].

En els següents accessos al servei només serà necessari que el *user* presenti un ${\mathsf{R}^*}_{\xi_i}$ vàlid i la data actual és abans de la LIMdate.

CAPÍTOL 4

IMPLEMENTACIÓ

La implementació es dividirà en tres etapes. A la primera etapa, s'implementaran tots els actors dins el mateix servidor i accedint als mateixos recursos amb l'objectiu de mostrar el correcte funcionament del protocol. A continuació, es desplaçarà el *user* a un dispositiu mòbil i es mesuraran el temps d'execució del protocol per a decidir si el temps d'execució del protocol és prou ràpid per a poder implementar-ho en una situació real. En la darrera etapa, s'implementarà un prototip del sistema real.

4.1 Decisions d'implementació

Davant de la complexitat del protocol, s'ha decidit dividir la implementació del protocol en diferents fases per a simplificar aquest procés. Aquesta aproximació permetrà dividir la implementació i enfocar cada una de les fases a resoldre un problema concret del projecte.

Les tecnologies que s'empraran s'elegiran amb l'objectiu d'adaptar-se a les darreres tendències en el món professional. Amb aquesta mentalitat s'ha descartat emprar JAVA directament per a crear un Web application ARchive (WAR) i desplegar-ho a una plataforma Tomcat, ja que aquesta tecnologia no suporta gaire bé canvis menors en el codi, allargant el procés d'implementació. En el seu lloc s'emprarà *SpringBoot* un *framework* de JAVA que permet crear aplicacions d'*String* simplificant el procés de configuració. El resultat d'aquest *framework* són aplicacions de qualitat de producció sense la necessitat d'un llarg procés de configuració.

Les opcions contemplades a l'hora de crear aquest sistema s'han limitat a solucions basades en JAVA. Això permetrà reutilitzar el codi en el moment de crear l'aplicació mòbil per a ANDROID. Aquesta restricció ha suposat que no s'han considerat les següents opcions:

DJANGO: *Framework* de PYTHON orientat a aplicacions web que ofereix un sistema ràpid, segur i escalable per a desenvolupar projectes sense la necessitat d'entrar a la configuració d'alt nivell.

RUBY ON RAILS: *Framework* per a crear aplicacions web basades en RUBY basat en dos principis: *Don't repeat yourself* (intenta evitar la repetició de codi redundant) i *Convention-over-configuration* (configuració automàtica del sistema per a facilitar les tasques del programador).

ASP.NET: *Framework* desenvolupat per MICROSOFT per al desenvolupament d'aplicacions web emprant qualsevol llenguatge web suportat per .NET (com per exemple C++, C# o F#). Aquest sistema està especialment dissenyat per a executar-se en màquines emprant el Sistema Operatiu (SO) WINDOWS.

NODE.JS: Sistema desenvolupat per a executar codi JAVASCRIPT en el costat del servidor. Tot i a ser un dels més recents, la seva popularitat ha augmentat de manera considerable en els darrers anys.

Tot i que aquestes plataformes ofereixen opcions molt interessants, la familiaritat amb el llenguatge de programació JAVA, la quantitat de documentació disponible i la possibilitat de reutilitzar codi han estat motius suficients per a la decisió final sobre quin sistema s'emprarà per a implementar el projecte.

A l'hora d'elegir la base de dades a emprar s'han contemplat les opcions següents:

SQL: Les bases de dades SQL emmagatzemen la informació en taules, requerint un esquema que les defineixi abans de poder-les emprar. Totes les entrades dins aquesta taula hauran de complir amb aquest esquema. Permeten l'execució de JOINs per a accedir a informació de diverses taules emprant un sola comanda [13] .

NOSQL: Les bases de dades NOSQL emmagatzemen la informació en fitxers *name-value* en format JSON. No suporten (ni haurien de ser necessaris) l'execució de JOINs. Permeten emmagatzemar informació sense especificar un esquema i permeten emmagatzemar informació en qualsevol moment a qualsevol posició sense cap verificació. Milloren l'escalabilitat del sistema [13].

Tot i que les bases de dades NOSQL han augmentat la seva popularitat en els darrers anys, s'ha decidit implementar una base de dades SQL ja que serà necessari relacionar el contingut de diverses col·leccions per a la correcte execució del sistema.

L'aplicació per a dispositius mòbils es desenvoluparà per a ANDROID. Aquest sistema operatiu ofereix la possibilitat de crear una aplicació per a la majoria dels *smartphones* en circulació avui [14] emprant el popular llenguatge de programació JAVA.

S'ha descartat el desenvolupament d'una aplicació per IOS perquè és necessari emprar un dispositiu amb el SO OSX. A més a més, les aplicacions per a aquesta plataforma han d'estar escrites en OBJECTIVE-C/C/C++ o SWIFT, eliminant la possibilitat de reutilitzar codi. Cal destacar també que les restriccions a l'hora d'accedir al xip NFC del dispositiu suposen una limitació per a futures revisions del projecte.

Tot i així, les eleccions de tecnologies actuals permetrien una adaptació futura del sistema per a suportar aquest SO.

No s'ha contemplat el desenvolupament per a qualsevol altre SO mòbil ja que cap d'ells disposa d'una quota de mercat que justifiqui invertir en aquest sistema.

També s'ha de mencionar l'ús de GIT per a mantenir un control de versions (així com una còpia de seguretat). S'ha decidit dividir la programació en diferents *branques* per a

cada una de les funcionalitats que s'implementaran del sistema. Aquest enfocament ens permetrà desenvolupar noves funcionalitats sense afectar al codi ja existent.

4.1.1 Sistema d'encriptació emprat

Durant la definició del protocol es menciona l'ús d'un algoritme de clau pública. Aquest algoritme s'empra principalment per a signar el contingut dels missatges amb l'objectiu d'assolir el requeriment de seguretat de no rebuig en origen. També s'empra per a encriptar informació crítica que només ha de ser visible per a alguna de les parts, com el TTP o el *provider*.

Tot i així, no s'especifica quin algoritme s'emprarà durant la implementació del protocol. Per això, s'han considerat:

RSA: Un dels sistemes de criptograma de clau pública més antics, aquest algoritme basa la seva seguretat en la dificultat pràctica de factoritzar el producte de dos primers grans. Aquest sistema permet tant encriptar com signar informació. És d'ús públic des de l'any 2000.

ElGamal: Aquest algorisme d'encriptació de clau pública està basat en l'intercanvi de claus *Diffie-Hellman*. Aquest sistema va augmentar la seva popularitat durant els anys 90 gràcies a que era de domini d'ús públic a diferència de **RSA**. Tot i així, ha perdut bastanta popularitat des de que **RSA** va entrar dins el domini públic.

DSA: Algorisme desenvolupat pel govern dels Estats Units per a la firma d'informació. Es sol emprar juntament amb el sistema d'encriptació **ElGamal**.

El sistema d'encriptació elegit és **RSA** degut a la seva popularitat i l'existència de llibreries molt eficients per a JAVA.

La repartició de claus es realitzarà mitjançant certificats. Aquests estaran basats en l'estàndard `X.509` desenvolupat per l'ITU-T i orientats a una infraestructura de clau pública. Aquest protocol assumeix un sistema jeràrquic estricte de Certificate Authority (CA). En aquest sistema els usuaris estan validats per CA intermedis certificats. Aquests CA estan certificats per altres entitats fins a arribar a una àncora de confiança. `X.509` va ser publicat el 1988 i ha esdevingut la base de molts protocols de seguretat d'Internet, com per exemple el protocol TLS/SSL emprat per a les connexions segures HTTPS [15].

L'estructura d'aquests certificats és la següent:

- Certificat:
  - Versió.
  - Número de serie del certificat.
  - ID de l'algorisme emprat pel CA per firmar (típicament **RSA** o **DSA**).
  - Emissor (CA).
  - Validesa:
    - No és vàlid abans de
    - No és vàlid després de

  - Subjecte:
    - *Common Name* (CN).
    - *Organizational Unit* (OU).
    - *Organization* (O).
    - *Country* (C). Aquest subjecte pot ser una persona, un servidor o un servei.
  - Informació de clau pública del subjecte:
    - Algoritme de clau pública.
    - Clau pública del subjecte.
  - Altres camps opcionals.

Per a crear els certificats, s'ha emprat OPENSSL. Aquest *open-source software* escrit en C implementa les funcions criptogràfiques bàsiques així com altres funcions addicionals. Una d'aquestes funcions és la de crear certificats **RSA**, així com la de signar un certificat. Aquesta darrera característica ens permetrà crear la nostra pròpia cadena de certificats amb el TTP com a CA.

Per a crear els certificats s'han emprat les comandes següents:

```
1.- openssl genrsa -out keypair.pem 2048

2.- openssl rsa -in keypair.pem -outform DER -pubout -out public.der

3.- openssl pkcs8 -topk8 -nocrypt -in keypair.per
 -outform DER -out private.der

4.- openssl req -new -key keypair.pem -out server.csr

5.- openssl x509 -req -days 365 -in server.csr
-signkey keypair.pem -out server.crt
```

El resultat d'aquest procés serà un certificat signat en aquest cas per ell mateix, però es pot emprar un altre certificat. Aquesta característica permetria signar tots els certificats dels diferents actors amb el certificat del TTP. D'aquesta manera obtindrem un certificat `X.509` i una clau privada que el nostre codi escrit en JAVA podrà llegir per a realitzar les operacions criptogràfiques.

4.2 Relació entre les entitats

A l'hora d'implementar el protocol s'hauran de considerar les relacions entre les diferents entitats. Aquestes relacions queden definides a la Figura 4.1.

- Un *provider* dóna d'alta els serveis que oferirà al *framework* comunicant-se amb l'*Issuer*.
- Un *user* generarà un nou pseudònim comunicant-se amb el TTP.
- Un *user* comprarà un PASS comunicant-se amb l'*Issuer*.

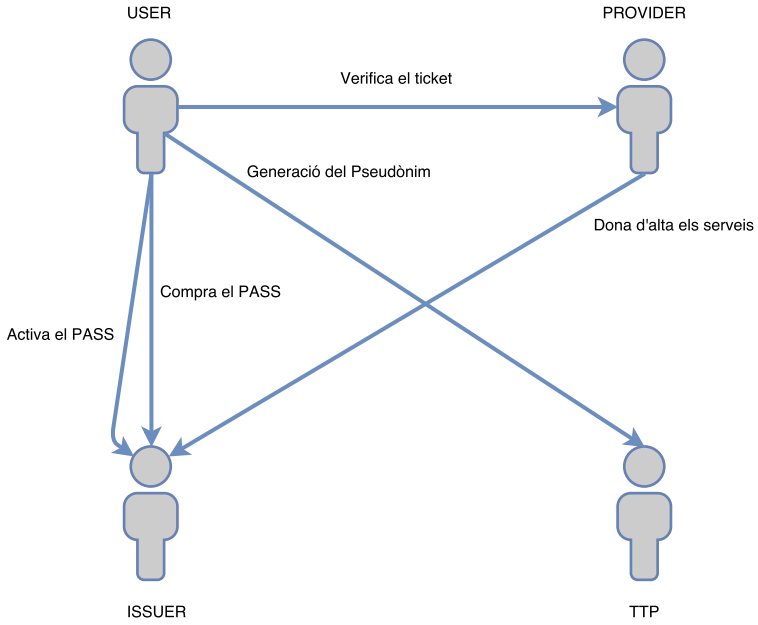

Figura 4.1: Diagrama de la relació entre les entitats

- Un *user* activarà un PASS comunicant-se amb l'*Issuer*.
- Un *user* verificarà un *ticket* comunicant-se amb un *provider*.

Les diferents etapes de la implementació es dividiran segons el mètode que s'emprarà per a realitzar les comunicacions entre les diferents entitats. L'arquitectura del sistema que s'emprarà dependrà de l'objectiu que es vol assolir a cada una de les etapes.

4.3 Disseny de la base de dades

El disseny de la base de dades s'ha realitzat una vegada s'ha definit la informació corresponent a cada actor, així com altres paràmetres del sistema i la relació entre ells.

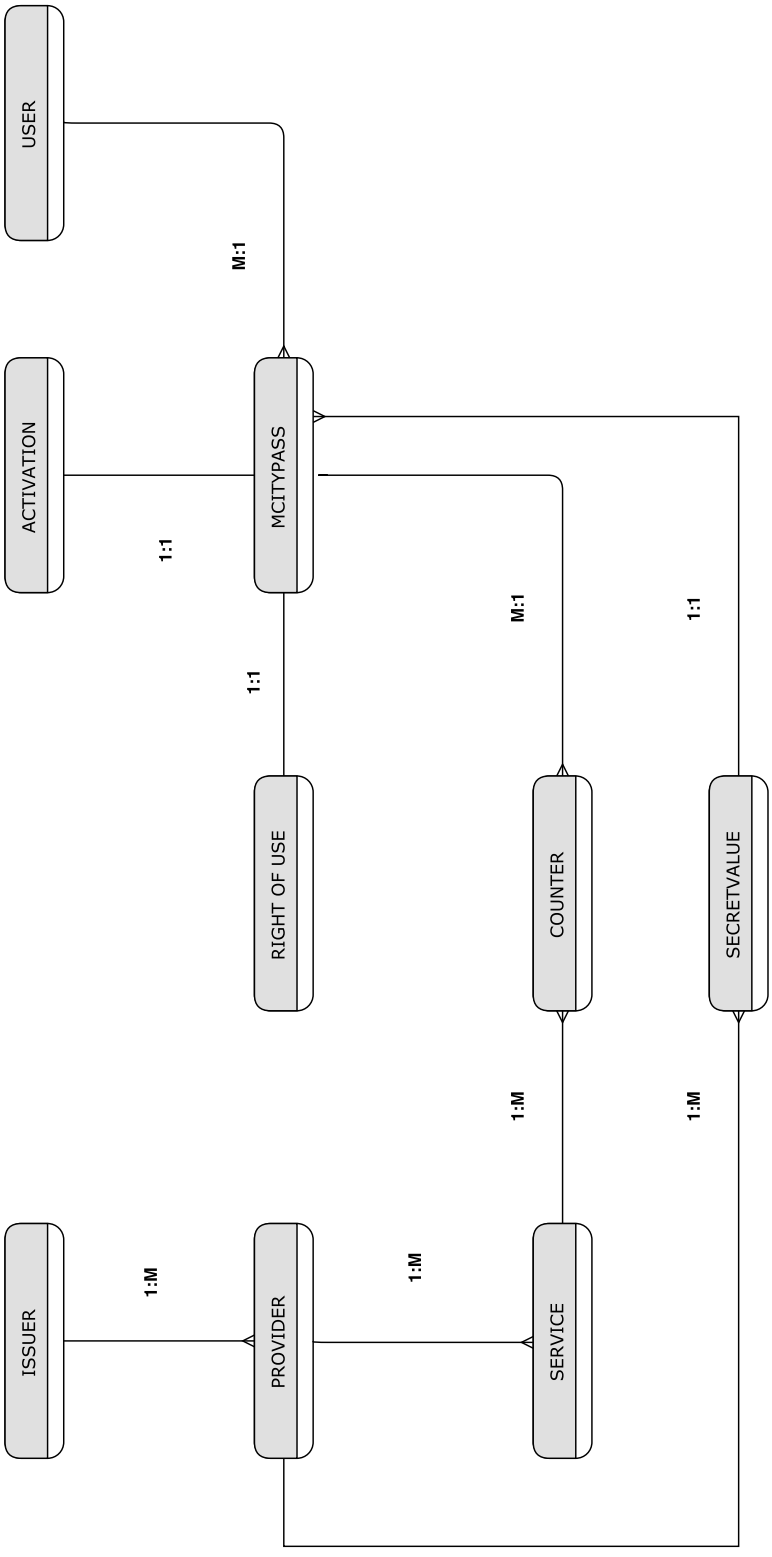

Figura 4.2: Diagrama de la Base de Dades durant la primera fase de la implementació

Com es pot observar a la Figura 4.2, la base de dades està dividida en deu taules. Cada una d'aquestes taules emmagatzema la informació corresponent a:

User: Informació corresponent a tots els usuaris donats d'alta dins el sistema. Aquesta informació fa referència al seu pseudònim i no a la seva identitat real, que només coneixerà el TTP en un procés no definit dins aquest protocol.

Issuer: Informació corresponent als *issuers* del sistema. Durant aquesta implementació es suposarà que només n'hi haurà un.

Provider: Informació corresponent als *providers* del sistema. És important indicar que un d'aquests paràmetres és l'*Issuer* al qual s'han afiliat. Tots els *providers* estan associats a un *issuer*.

TTP: Informació corresponent al TTP.

Serveis: Informació corresponent a cada un dels serveis que ofereix el sistema. Tots els serveis estan associats a un *provider*.

MCityPass: Informació corresponent a cada un dels PASS creats pel sistema. Tots els PASS estan associats a un *user* i tants de *Counters* com serveis hi ha dins el PASS.

Activation: Informació corresponent a la activació dels PASS. Cada *Activation* està relacionat a un PASS.

Counter: Guarda la informació sobre quantes vegades s'ha emprat un *ticket* d'un PASS per a evitar l'*overspending* del servei. Cada *Counter* està relacionat amb un servei i un PASS.

RightOfUse: Informació generada durant la creació del PASS que permet al *Provider* i al TTP verificar que el *user* intentant emprar un *ticket* n'és el propietari. Cada *RightOfUse* està relacionat amb un PASS.

SecretValue: Valor generat durant la creació del PASS que permet al *user* indicar que n'és el propietari alhora d'accedir a un servei. Cada *SecretValue* està relacionat amb un PASS i a cada *provider* que ofereix un servei dins aquell PASS.

El disseny de la base de dades que s'executarà al servidor a la segona fase de la implementació serà el mateix, eliminant només les taules que només pot conèixer l'usuari com el *SecretValue*. Aquest nou disseny es pot observar a la Figura 4.3.

Aquests valors que només pot conèixer el *user* seran emmagatzemats per ell al seu dispositiu mòbil.

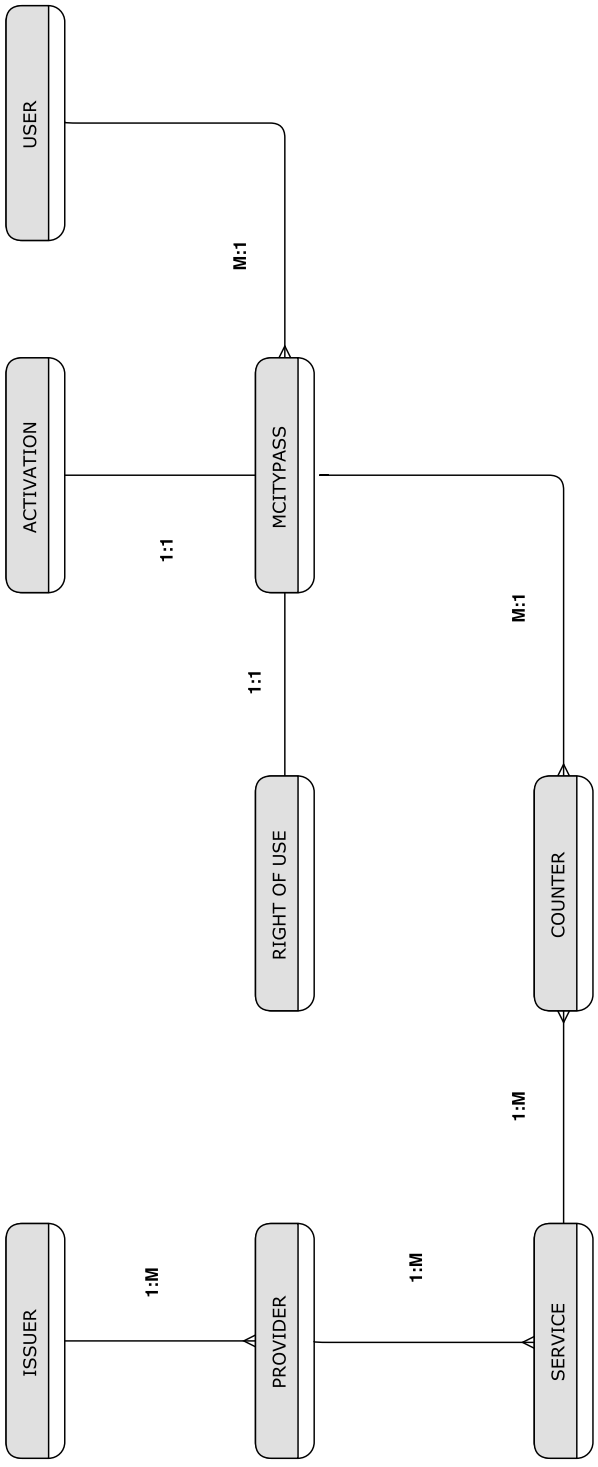

Figura 4.3: Diagrama de la Base de Dades durant la segona fase de la implementació

4.4 Primera etapa: Implementació del protocol

La primera etapa es centrarà en demostrar que el protocol és implementable i realitzar els ajustaments necessaris al disseny per a poder aconseguir-ho.

4.4.1 Disseny de la implementació

En aquesta etapa de la implementació, tots els actors s'implementaran al servidor, on tots tendran accés a la base de dades. El disseny emprat queda definit a la Figura 4.4.

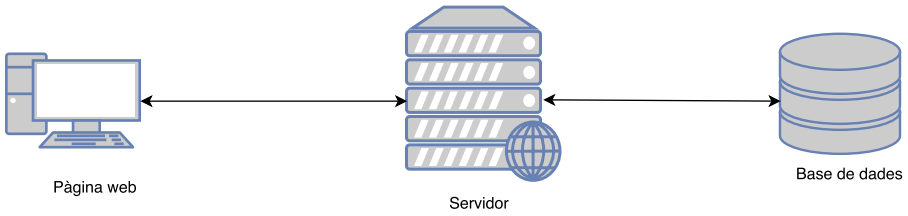

Figura 4.4: Diagrama de la primera etapa de la implementació

Per a iniciar l'execució d'una fase del protocol, s'accedirà a una plana web que mostrarà el resultat de la comunicació entre les diferents parts. Aquesta estructura permet concentrar-se en el correcte funcionament del protocol sense haver de configurar les comunicacions entre els actors.

4.4.2 Tecnologies emprades

La tecnologia en que es basarà la implementació és *Spring Boot*.

Spring és un *framework* de JAVA enfocat a gestionar la creació dels objectes mitjançant fitxers XML o *JavaBeans* emprant la Injecció de Dependències [16]. A més a més, també facilita la implementació d'aplicacions web basades en l'estructura Model-Vista-Controlador i que implementen serveis RESTful. Els serveis RESTful es caracteritzen per una arquitectura client servidor i estan dissenyats per emprar un protocol de comunicació *stateless* (com HyperText Transfer Protocol (HTTP)). En aquesta arquitectura l'intercanvi d'informació es realitza emprant una interfície i protocols predefinits [17].

Dins l' *Spring framework* la implementació emprarà el mòdul *Spring Boot*, que facilita la creació d'aplicacions *stand-alone* sense haver d'entrar a la configuració d'*Spring*. Aquest mòdul permet crear un servidor *Tomcat* sense la necessitat de desplegar fitxers WAR, automàticament configurant Spring sempre que sigui possible i oferint fitxers POM per simplificar la configuració de *Maven* (o *Gradle*) [18].

La *construcció* de l'aplicació JAVA es realitzarà emprant *Gradle* enlloc de *Maven*. Aquest sistema facilita la declaració de dependències gràcies a l'ús d'un llenguatge de programació de domini especific basat en GROOVY (llenguatge *script* per a la plataforma JAVA desenvolupat per Apache) en comparació amb el sistema basat en `XML` de *Maven* i *Ant*. A més a més, *Gradle* ofereix més flexibilitat alhora de compilar el projecte, reconeixent quines parts del *build tree* estan actualitzades i només compilant les parts necessàries [19]. Aquest sistema és l'adoptat per Google per a la construcció d'aplicacions per a Android.

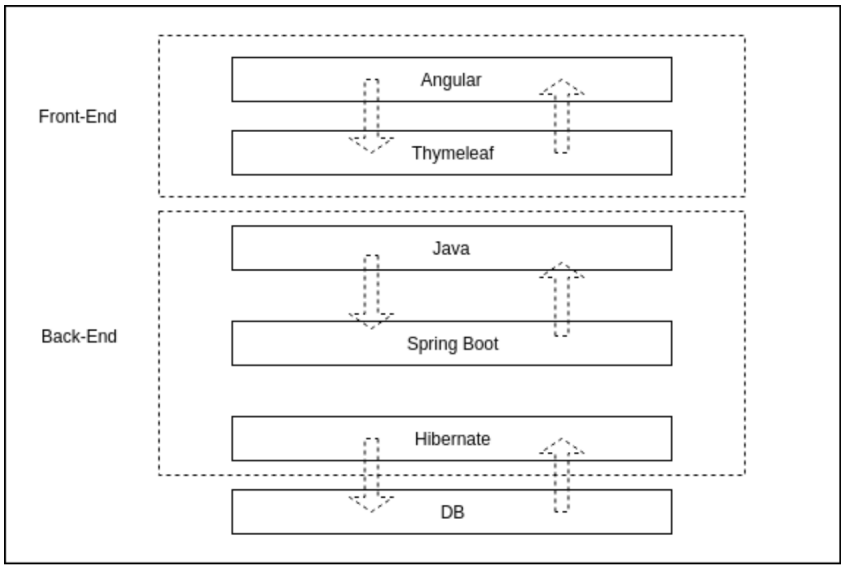

Figura 4.5: Estructura de les tecnologies emprades.

L'estructura emprada per a implementar l'aplicació és la definida a la Figura 4.5 . Es pot observar com el codi s'implementarà en JAVA, sobre les eines que ofereix *Spring Boot*. A més a més, s'emprarà *Hibernate* per a realitzar la traducció entre les classes del codi JAVA i les taules de la base de dades. Aquest *framework* evita al programador haver d'executar les consultes directament a la base de dades i executa la traducció d'objectes (i els tipus d'informació).

Les operacions matemàtiques que es realitzaran durant l'execució s'han implementat emprant la classe BIGINTEGER. Aquesta classe permet crear enters immutables de precisió arbitrària, alhora que ofereix mètodes per a la creació de primers, aritmètica modular i operacions binàries [20]. Aquests mètodes simplificaran molt la implementació de les operacions corresponents a l'*Schnorr's* ZKP i l'ús d'un PRNG (*Pseudo-Random Number Generator*).

La base de dades emprada és HSQLDB (*Hyper SQL DataBase)*. Aquesta base de dades ofereix un motor senzill i ràpid, alhora que el fet de que estigui implementat en JAVA ens permet garantir que es podrà emprar en tots els dispositius on s'executarà el servidor. Tot i així, la característica que l'ha situada com la millor opció per al desenvolupament del projecte és que permet executar-se en memòria, és a dir, la base de dades es crea cada vegada que s'executa el programa [21]. Això permet:

- Eliminar la necessitat d'instal·lar *software* addicional per a desenvolupar el projecte.
- Compartir els arxius de configuració entre els diferents membres de l'equip per a què tots puguin treballar sobre la mateixa base de dades.
- Tornar a estats anteriors de la base de dades restaurant els arxius de configuració.

A més a més, l'ús del llenguatge SQL permetrà migrar les taules i el seu contingut a una plataforma més permanent a l'hora d'iniciar la fase de producció del projecte.

Els accessos a la base de dades es realitzaran emprant un Object-relational Mapping (ORM). Aquest *software* crea una "base de dades d'objectes virtual"que amaga els accessos a la base de dades, podent guardar i extreure elements de manera transparent. D'aquesta manera, el programador no ha de realitzar consultes SQL. Alhora d'implementar aquest projecte s'ha emprat HIBERNATE, un *framework* per a JAVA desenvolupat per Red Hat. Aquest ORM permet crear les bases de dades de manera automàtica a partir del sistema d'anotacions de JAVA podent especificar les relacions entre les taules [22].

El *frontend* s'implementarà emprant *Thymeleaf* com a alternativa a JSP (*JavaServer Pages*). El seu objectiu és oferir *natural templates*, és a dir, HTML que es pot mostrar correctament a un navegador i també funcionar com prototips estàtics, permetent un nivell de col·laboració molt més alt [23]. Aquesta llibreria permet crear plantilles de manera senzilla, especialitzant-se en la capa de Vista de les aplicacions web basades en MVC (Model-Vista-Controlador). Aquesta arquitectura de disseny de *software* per a interfícies gràfiques divideix l'aplicació en tres parts interconnectades on cada una realitza una tasca diferent [24]:

- Model: Conté la lògica del programa, independentment de la interfície gràfica.
- Vista: Mostra la informació a l'usuari.
- Controlador: Realitza les traduccions entre el model i la vista.

4.4.3 Implementació

La implementació s'ha realitzat emprant el paradigma de disseny Facade-Servei-Model. Aquest paradigma diferencia tres capes a l'hora de dissenyar el codi [25] :

Facade: És la capa més alta del codi, oferint un punt d'accés als clients separant-los dels serveis.

Servei: Implementa la lògica del programa.

Model: Fa referència a l'estructura de dades emprada, així com a la persistència de dades.

Tot i que el paradigma realitza defineix aquests tres tipus d'entitats, no s'implementaran els mètodes corresponents als *facades* fins que no es decideixi per a una implementació definitiva del projecte ja que aquests defineixen els punts d'accés dels usuaris. Així doncs, aquesta tasca variarà segons com es decideixi realitzar la implementació final i, en aquesta etapa del desenvolupament, no oferirien cap funcionalitat que justificàs l'augment en complexitat. Tot i així, el codi s'ha preparat per a la seva implementació.

A més a més, s'han creat DAOs (*Data Access Objects*) per a separar la lògica del programa de l'accés a la base de dades. Aquestes classes defineixen objectes que ofereixen interfícies abstractes que permeten realitzar operacions amb les dades sense mostrar informació sobre aquesta [26].

També cal destacar la creació de dos paquets amb funcionalitats molt definides:

- *crypto:* Defineix totes les operacions criptogràfiques que empraran els diferents actors. Inclou tant les operacions de *Schnorr's ZKP* com l'encriptació i firma emprant RSA. Per a l'ús de *Schnorr's ZKP* s'han implementat totes les operacions matemàtiques necessàries per a dur a terme aquest protocol. Les operacions criptogràfiques que empren RSA usaran les classes oferides pel paquet `java.security`. S'ha de mencionar que per a la codificació en BASE64 s'ha emprat la classe oferida per GOOGLE per al desenvolupament per a ANDROID degut a problemes de compatibilitat entre aquest i els mètodes oferits pel paquet `java.util`.
- *utils:* Defineix mètodes i valors estàtics que empraran els diferent actors.

Models

A continuació s'explicaran els models implementats:

- *Activation*: Defineix els paràmetres continguts dins l'entitat *Activation*, així com els seus constructors. Aquests valors inclouen:
  - El PASS al qual fa referència.
  - La data d'activació.
  - L'estat del PASS.
- *Counter*: Defineix els paràmetres continguts dins l'entitat *Counter*, així com els seus constructors.
  - El PASS al qual fa referència.
  - El servei al qual dona accés.
  - El nombre restant de vegades que es pot emprar el servei. Si es tracta d'un servei infinitament reutilitzable contarà el nombre de vegades que s'hi ha accedit.
  - El darrer valor emprat per a accedir al servei.
  - El valor calculat pel *user* durant la fase de Compra del PASS com a darrer valor de la cadena de *hash*.
- *Issuer*: Indica els paràmetres que defineixen un *Issuer*. Aquests paràmetres són:
  - Nom de l'*Issuer*.
  - Llista dels *providers* que té associats.
- *MCityPass*: Defineix els paràmetres continguts dins l'entitat *MCityPass*, així com els seus constructors. Aquests paràmetres són:
  - El *user* propietari del PASS.
  - La llista de *Counters* que tenen associats aquest PASS.
  - L'*Activation* d'aquest PASS.
  - El *RightOfUse* associat a aquest PASS.

  - El temps de vida d'aquest PASS.
  - La categoria d'aquest PASS.
  - Els termes i condicions d'aquest PASS.
  - Altres valors generats durant la compra del PASS necessaris per a accedir a un servei.

- *Provider*: Indica els paràmetres que defineixen un *Provider* dins el protocol. Aquests paràmetres són:
  - El nom del *Provider*.
  - L'*Issuer* al qual està associat.
  - La llista de serveis que ofereix.

- *RightOfUse*: Defineix els paràmetres continguts dins l'entitat *RightOfUse*, així com els constructors. Aquests paràmetres són:
  - El PASS al qual està associat.
  - La clau de sessió generat durant la compra del PASS.
  - La signatura sobre aquesta clau de sessió.

- *SecretValue*: Defineix els paràmetres continguts dins l'entitat *SecretValue*, així com els seus constructors. Aquests valors són:
  - El *Provider* què ofereix aquest servei.
  - El PASS al qual està associat aquest valor.
  - El valor generat durant la compra del PASS.

- *ServiceAgent*: Defineix els paràmetres continguts dins l'entitat *ServiceAgent*, així com els seus constructors. Aquests valors són:
  - El nom d'aquest servei.
  - El *Provider* què ofereix aquest servei.
  - El nombre de vegades que es pot emprar el servei. Si aquest servei és infinitament reutilitzable aquest valor serà -1.
  - La llista de comptadors que fan referència a aquest servei.
  - Altres valors generats durant l'afiliació del *Provider*.

- *TrustedThirdParty*: Defineix els paràmetres continguts dins l'entitat *TrustedThirdParty*, així com els seus constructors. Aquests valors són:
  - El nom del *TrustedThirdParty*.

- *User*: Indica els paràmetres que defineixen aquest actor del protocol. Aquests valors són:
  - El pseudònim..
  - Els valors de l'*Schnorr's ZKP* corresponents a aquest usuari.
  - La llista de PASS que ha comprat.

Serveis

A continuació s'explicaran els serveis implementats:

- *ActivationService*: Conté tota la lògica corresponent a les entitats *Activation*. Principalment, aquesta classe conté mètodes d'alt nivell per a accedir a entitats de la base de dades. Aquest mètodes permeten a qualsevol altre servei que vulgui accedir a un *Activation* accedir al contingut corresponent de la base de dades.

- *CounterService*: Conté tota la lògica corresponent a l'entitat *Counter*. Principalment, aquesta classe conté mètodes d'alt nivell per a accedir a entitats de la base de dades. Aquest mètodes permeten a qualsevol altre servei que vulgui accedir a un *Counter* accedir al contingut corresponent de la base de dades.

- *IssuerService* Conté tota la lògica corresponent a l'entitat *Issuer*. Aquesta classe inclou tots els mètodes necessaris per a executar les diferents fases del protocol on aquest actor està involucrat:

  – Afiliació dels *providers*.

  – Compra d'un PASS per part d'un *user*.

  – Activació d'un PASS per part d'un *user*.

  A més a més, també s'inclouen tots els mètodes d'alt nivell per a accedir a entitats de la base de dades.

- *MCityPassService*: Conté tota la lògica corresponent a l'entitat *MCityPass*. Principalment, aquesta classe conté mètodes d'alt nivell per a accedir a entitats de la base de dades. Aquest mètodes permeten a qualsevol altre servei que vulgui accedir a un *MCityPass* accedir al contingut corresponent de la base de dades. A més a més, també s'han declarat mètodes per a verificar la validesa d'un *MCityPass*.

- *ProviderService*: Conté tota la lògica corresponent a l'entitat *Provider*. Aquesta classe inclou tots els mètodes necessaris per a executar les diferents fases del protocol on aquest actor està involucrat:

  – Afiliació dels *providers*.

  – Verificació d'un *ticket* per part d'un *user*.

  A més a més, també s'inclouen tots els mètodes d'alt nivell per a accedir a entitats de la base de dades.

- *RightOfUseService*: Conté tota la lògica corresponent a l' entitat *RightOfUse*. Principalment, aquesta classe conté mètodes d'alt nivell per a accedir a entitats de la base de dades. Aquest mètodes permeten a qualsevol altre servei que vulgui accedir a un *RightOfUse* accedir al contingut corresponent de la base de dades.

- *SecretValueService*: Conté tota la lògica corresponent a l' entitat *SecretValue*. Principalment, aquesta classe conté mètodes d'alt nivell per a accedir a entitats de la base de dades. Aquest mètodes permeten a qualsevol altre servei que vulgui accedir a un *SecretValue* accedir al contingut corresponent de la base de dades.

- *ServiceAgentService*: Conté tota la lògica corresponent a l'entitat *ServiceAgent*. Principalment, aquesta classe conté mètodes d'alt nivell per a accedir a entitats de la base de dades. Aquest mètodes permeten a qualsevol altre servei que vulgui accedir a un *ServiceAgent* accedir al contingut corresponent de la base de dades.

- *TrustedThirdPartyService*: Conté tota la lògica corresponent a l'entitat TTP. Aquesta classe inclou tots els mètodes necessaris per a executar les diferents fases del protocol on aquest actor està involucrat, especialment la fase de generació del pseudònim per part de l'usuari.

- *UserService*: Conté tota la lògica corresponent a l'entitat *User*. Aquesta classe inclou tots els mètodes necessaris per executar les diferents fases del protocol on aquest actor està involucrat:

  - Negociar amb el TTP la generació d'un pseudònim.
  - Negociar la compra d'un PASS amb un *Issuer*.
  - Negociar l'activació d'un PASS amb un *Issuer*.
  - Demostrar a un *provider* la validesa d'un *ticket*.

  A més a més, també s'inclouen tots els mètodes d'alt nivell per a accedir a entitats de la base de dades.

Controladors

- *HomePageController*: L'accés a aquest controlador des d'un navegador inicia l'execució del protocol i mostra per pantalla si s'han executat totes les fases correctament. Els paràmetres per a l'execució del protocol s'han definit dins aquest protocol.

4.4.4 Conclusió

Una vegada acabada la implementació del protocol amb resultats favorables (discutits a la Secció 5.1), es començarà la segona etapa de la implementació.

4.5 Segona etapa: Mesures temporals de la fase de Verificació del PASS

La segona etapa es centrarà en desplaçar el *user* a un altre dispositiu per a realitzar les mesures del temps que tarda en executar-se la fase de verificació (Secció 3.3.5). Les mesures temporals es centraran en aquesta fase perquè és la més sensible alhora d'implementar el protocol en un sistema real, ja que podria augmentar les coes per a accedir a un servei fins al punt de suposar què el protocol no és viable. Les altres fases no són tan crucials temporalment perquè es poden executar en qualsevol moment i no en el moment en què el *user* vol accedir a un servei.

4.5.1 Disseny de la implementació

En aquesta etapa de la implementació, el *user* s'executarà en un dispositiu mòbil mentre que el *provider* s'executarà al servidor. El disseny emprat queda definit a la Figura 4.6.

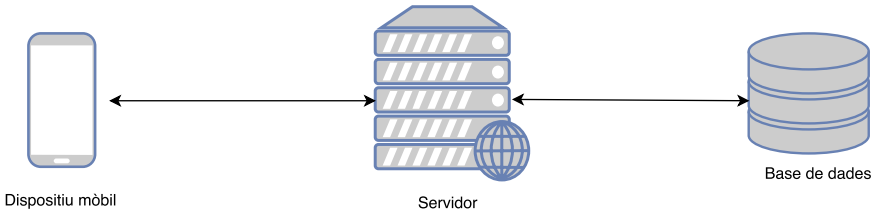

Figura 4.6: Diagrama de la segona etapa de la implementació

Per a executar la verificació d'un *ticket*, el *user* accedirà a una API que defineix totes les funcions que executa el *provider* mitjançant una connexió HTTP.

En aquest cas el dispositiu mòbil estarà connectat a una xarxa Wireless Local Area Network (WLAN) connectada a la mateixa xarxa cablejada que el servidor.

4.5.2 Tecnologies emprades

L'aplicació mòbil es programarà per a dispositius Android emprant JAVA. Android és un SO per a dispositius mòbils desenvolupat per Google basat en el *kernel* Linux. Actualment aquest SO domina el mercat d'*smartphones*, executant-se en un 82% del mercat global [14] i un 92.9% del mercat espanyol [27] el 2016. Una de les raons de la popularitat d'aquesta plataforma és la possibilitat d'adquirir dispositius de diferents fabricants què empren aquest SO, oferint una gran varietat de dispositius disponibles de diferents preus i característiques.

La comunicació entre el dispositiu mòbil i el servidor on s'executarà el *provider* es realitzarà emprant una connexió HTTP.

Cal destacar l'ús de la llibreria RETROFIT desenvolupada per SQUARE, que permet convertir una API HTTP en una interfície de JAVA [28]. Aquesta llibreria simplifica molt les connexions al servidor, amagant les connexions HTTP al programador i permetent obtenir JavaScript Object Notation (JSON) de resposta i convertir-los a objectes de JAVA de manera automàtica.

4.5.3 Implementació

La implementació d'aquesta fase s'ha centrat en separar l'execució del *user* del servidor. A la primera etapa per a aconseguir-ho s'ha preparat el codi per executar-se al dispositiu mòbil. Per a fer-ho s'han realitzat les següents tasques:

- Eliminar els *beans* emprats per *Spring*.
- Eliminar els accessos a la base de dades i *hard coding* els valors que s'empraran per a la validació.

Per a simplificar la implementació, els valors que emprarà el *user* per a realitzar les comprovacions han estat *hard coded*. Aquesta pràctica emprada durant l'etapa del

desenvolupament de *software* consisteix en introduir els valors d'entrada d'un sistema directament al codi enlloc d'obtenir-los de manera dinàmica o d'una font externa [29]. Això permetrà agilitzar la implementació d'aquesta fase sense alterar la validesa dels resultats. En una futura etapa, es modificarà el codi per a obtenir tots els valors de manera dinàmica i obtenir un prototip més proper al producte final.

Una vegada s'ha comprovat que la validació segueix sent correcte, s'han traslladat les classes corresponents a la implementació del *user* al dispositiu mòbil. Aquest codi inclou tots els models, els serveis corresponents al *user* i els models que no són actors, i les classes de criptografia i utilitats.

A continuació, s'han implementat les connexions HTTP emprant la llibreria RETROFIT. Amb aquesta llibreria, cada un dels missatges intercanviats entre el *user* i el *provider* durant aquesta fase queda definit per un mètode amb la forma següent:

```
@POST("provider/verifyPass")
Call<String> verifyPass(@Body String json);
```

Per a realitzar aquestes consultes serà necessari crear una ASYNCTASK abans de realitzar les connexions ja que Android no permet realitzar connexions al *thread* principal. Des d'aquesta ASYNCTASK s'executarà la connexió emprant el següent codi:

```
ProviderAPI providerAPI = new Retrofit.Builder()
    .baseUrl(BASE_URL)
    .addConverterFactory(ScalarsConverterFactory.create())
    .build()
    .create(ProviderAPI.class);

Call<String> stringCall =
 providerAPI.verifyPass(userService.showTicket(1, 2));
```

Com es pot veure als fragments de codi anterior, RETROFIT ens permet executar les connexions des del *user* al *provider* sense haver de configurar els missatges HTTP.

D'altra banda, al servidor s'ha creat un controlador que realitza la funció d'API, permetent al *user* connectant-se als serveis del *provider*. Cada un d'aquest mètodes escoltarà a un Uniform Resource Locator (URL) diferent, i respondrà amb la resposta del *provider* corresponent al *query* del *user*. La forma que prendran aquests mètodes es pot observar al codi a continuació.

```
@RequestMapping(value = "/provider/verifyPass",
     method=RequestMethod.POST)
@ResponseBody
public String verifyPass(@RequestBody String json){
     return providerService.verifyPass(json);
}
```

4.5.4 Conclusió

En aquesta etapa del desenvolupament s'ha creat una primera aplicació per a un dispositiu mòbil amb l'objectiu d'obtenir el temps d'execució de la fase de verificació

del PASS (Secció 3.3.5). La metodologia emprada i els resultats obtinguts es discutiran a la Secció 5.2.

Obtingudes les mesures temporals, es continuarà la implementació desenvolupant una aplicació que permetrà a l'usuari disposar de les funcionalitats completes del protocol.

4.6 Tercera etapa: Desenvolupament d'una aplicació funcional

A la darrera fase s'ha desenvolupat una aplicació capaç només d'executar la fase de Verificació del PASS. Aquesta aplicació no permetia a l'usuari executar cap de les fases anteriors ni emmagatzemava informació de manera persistent. En aquesta etapa s'ha desenvolupat un prototip del sistema final, amb una aplicació que permet a l'usuari generar un pseudònim, comprar i activar diferents PASS i verificar un *ticket* per a accedir a un servei.

4.6.1 Disseny de la implementació

En aquesta etapa de la implementació, el *user* s'executarà en un dispositiu mòbil mentre que el *provider*, l'*Issuer* i el TTP s'executaran en el servidor. El disseny emprat queda definit a la Figura 4.7.

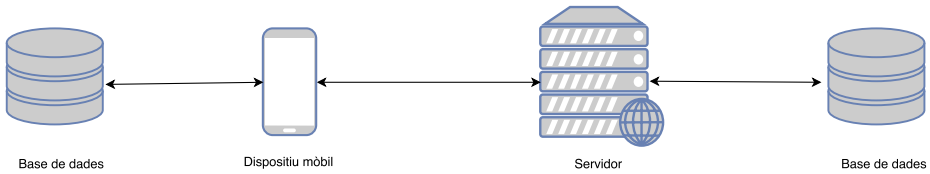

Figura 4.7: Diagrama de la tercera etapa de la implementació

Per a executar totes les fases del protocol, el *user* accedirà a una API que defineix totes les funcions executades pels altres actors mitjançant una connexió HTTP.

Cal destacar que en aquesta etapa el dispositiu mòbil disposarà de la seva pròpia base de dades, on s'emmagatzemarà tota la informació que necessitarà per a executar el protocol.

En aquest cas el dispositiu mòbil estarà connectat a una xarxa WLAN connectada a la mateixa xarxa cablejada que el servidor.

4.6.2 Tecnologies emprades

El desenvolupament de l'aplicació en aquesta etapa es basarà en la creada a l'apartat anterior. Així doncs, es faran servir les tecnologies que s'han definit a la Secció 4.5.2. Les úniques tecnologies afegides durant etapa són les referents a l'emmagatzematge d'informació.

Aquest emmagatzematge es realitzarà emprant una base de dades SQLITE. Aquesta motor de base de dades ofereix totes les funcionalitats oferides per SQL sense la necessitat de configurar-se ni d'executar un procés de servidor separat [30]. Aquesta

base de dades és la més popular en el mercat, ja que és present a la majoria de navegadors, tots els dispositius amb el SO IOS, tots els ordinadors Mac i Windows i, de major importància en el nostre cas, tots els dispositius ANDROID [31].

Aquesta llibreria permet crear una base de dades que s'executa dins l'aplicació, llegint i escrivint directament dins fitxers i amb la capacitat d'executar operacions ACID (*Atomic, Consistent, Isolated, Durable*) [30].

Tot i així, és recomanat realitzar els accessos a la base de dades de manera directe ja que programar cada una de les consultes SQL resulta una tasca molt tediosa i on és molt fàcil cometre errors.

Per a resoldre aquest problema s'ha emprat un ORM. Tot i que al servidor s'ha decidit usar *Hibernate* gràcies a la seva senzilla integració amb *SpringBoot*, aquest *framework* no està disponible alhora de programar per a ANDROID. Com a alternativa s'ha emprat ACTIVEANDROID, una llibreria *open-source* que s'ocupa de tota la configuració de la base de dades i permet realitzar consultes a la base de dades mitjançant codi JAVA molt senzill [32].

Per a crear una taula a la base de dades, només és necessari que aquest objecte estengui la classe *Model* i especificar quins camps es desitgen guardar. També permet indicar relacions *one-to-one*, *one-to-many* i *many-to-many*.

4.6.3 Implementació

La implementació d'aquesta etapa s'ha centrat en realitzar un prototip que permeti al *user* realitzar totes les fases del protocol. Amb aquest objectiu, el desenvolupament s'ha centrat en tres pilars principals:

- Emmagatzemar de manera persistent la informació.
- Configurar totes les comunicacions amb el servidor.
- Crear una interfície d'usuari.

Ja que a la primera etapa de la implementació tots els actors tenien accés a la mateixa base de dades i a la segona només s'executava una fase del protocol. La primera de les tasques ha estat definir quina informació necessitava emmagatzemar el *user* per a poder executar tot el protocol. Una vegada definida quina informació havia de guardar i quan, s'han realitzats els canvis corresponents al codi:

1. Tots els models estenen la classe *Model* de la llibreria *ActiveAndroid*. D'aquesta manera s'han definit quines taules haurà de tenir la nostra base de dades.
2. S'han indicat les columnes de cada taula emprant el sistema d'anotacions oferit per *ActiveAndroid* i s'han indicat les relacions entre els models.
3. S'han afegit mètodes als serveis per a realitzar les consultes a la base de dades. Degut a la simplicitat i estructura d'aquestes consultes s'ha decidit no crear DAOs com en el servidor.
4. S'han revisat els mètodes que implementen l'execució del protocol per modificar els accessos a la informació, tant per llegir com per emmagatzemar-ne.

A continuació s'han creat les interfícies corresponents als actors amb el qual haurà d'interactuar el *user* (*provider*, *issuer*, TTP) així com una per a accedir a la llista de serveis oferits. Aquestes interfícies s'han implementat emprant la llibreria RETROFIT. L'estructura d'aquestes interfícies s'ha definit a la Secció 4.5.3.

A més a més d'aquestes interfícies, s'ha creat una classe per a cada una de les interaccions del *user* amb el servidor. Aquestes estenen la classe ASYNCTASK, que permet realitzar les operacions de xarxa a un *thread* diferent al principal sense la necessitat de configurar *Handlers*.

Al servidor s'han implementat els *Controllers* següents:

IssuerController: Defineix totes les comunicacions entre el *Issuer* i el *User*. Ofereix mètodes per a la compra i verificació del PASS.

ProviderController: Defineix totes les comunicacions entre el *Provider* i el *User*. Ofereix mètodes per a la verificació del PASS.

TTPController: Defineix totes les comunicacions entre el TTP i el *User*. Ofereix mètodes per a la generació del PASS.

ServicesController: Defineix un mètode per a que el *User* pugui obtenir la llista de serveis oferits.

La interfície d'usuari de l'aplicació s'ha realitzat al voltant d'una sola *Activity*. Aquesta classe és el punt d'entrada d'interacció amb l'usuari i representa una sola pantalla amb una interfície gràfica [33]. Dins aquesta *Activity* es situaran diferents *Fragments* segons l'acció que desitgi realitzar l'usuari.

Aquest enfocament permet crear una interfície d'usuari molt ràpida, alhora que facilita modularitzar el codi i compartir informació entre les diferents tasques a realitzar. Cada un d'aquests *Fragments* disposa de la seva pròpia interfície gràfica (o *Layout*) corresponent a un arxiu `XML`.

4.6.4 Conclusió

Amb aquesta etapa finalitza la fase implementació d'aquest protocol. L'objectiu d'aquesta de la fase ha estat mostrar la viabilitat d'aquest protocol i mostrar un prototip d'una aplicació amb la qual els usuaris interaccionarien amb el *framework*.

A continuació es discutiran els resultats obtinguts durant aquesta fase de la implementació.

CAPÍTOL 5

JOCS DE PROVES

En aquest capítol es presentaran i discutiran els resultats obtinguts durant la implementació del protocol. Donat que la implementació s'ha dividit en tres etapes segons l'objectiu que es volia assolir, en aquesta etapa es realitzarà un anàlisi dels resultats obtinguts a cada una de les diferents etapes per separat.

5.1 Resultats de la primera etapa

L'objectiu d'aquesta primera etapa era demostrar que el protocol compleix amb els objectius que es proposa i compleix amb els requisits de seguretat. Per a obtenir el resultat s'han executat tan cada fase per separat com tot el protocol seguit, obtenint valors correctes a totes les comprovacions.

En aquesta fase els temps d'execució del protocol no ofereixen informació significativa ja que tots els actors s'estan executant al mateix dispositiu. Per tant, els valors temporals no presenten informació vàlida ja que el *user* s'està executant a un dispositiu amb més potència de càlcul del que s'executarà en el sistema real, alhora que no és té en compte el temps de transmissió dels missatges.

5.2 Anàlisi de rendiment

A continuació es mostraran els resultats de l'anàlisi de rendiment efectuat sobre el prototipus definit a la Secció 4.5.

5.2.1 Procediment per a obtenir els resultats

Per a obtenir els resultats s'ha creat una xarxa WLAN sense accés a Internet emprant un *router* amb capacitats d'Acces Point (AP). Els dispositius mòbils s'han connectat emprant la xarxa Wi-Fi. La funció del servidor l'ha realitzat l'ordinador on s'ha desenvo-

lupat la implementació del sistema. Aquest ordinador s'ha connectat a la xarxa emprant un dels ports de xarxa cablejada del *router* amb una connexió *Fast Ethernet*.

Alhora d'obtenir els resultats s'ha mesurat el temps d'execució del protocol des de l'inici de la comprovació des del mòbil fins que el *user* dóna per finalitzada la fase, imprimint per pantalla el *timestamp* quan comença i acaba l'execució.

Les mesures s'han realitzat cinc vegades per a cada dispositiu, efectuant la mitjana d'aquestes per a eliminar l'efecte de processos de segon pla executant-se en el dispositiu mòbil.

Els dispositius mòbils que s'han emprat són:

Samsung Galaxy Note 2: Aquest dispositiu va ser presentat el 2012 per Samsung, com un dels primers *phablets* dirigits al públic general. Amb un *hardware* de gama alta en el moment en què es va presentar i oferint funcionalitats addicionals gràcies a l'*S-Pen* va ser rebut amb gran acceptació pel públic venent més de 30 millons de dispositius.

Sony Xperia P: Aquest dispositiu va ser presentat el 2012 per Sony juntament amb Ericcson enfocat a la gama mitja. Tot i les seves característiques de gama mitja, cal destacar l'ús d'alumini per a la construcció i el seu suport d'NFC.

Xiaomi Mi4c: Aquest dispositiu va ser presentat el 2015 per Xiaomi, oferint característiques de dispositius de gama alta a un preu molt més baix. S'ha de mencionar que aquest dispositiu empra la capa de personalització MIUI desenvolupada per Xiaomi, caracteritzada per les seves actualitzacions setmanals.

Sony Xperia Z5 Compact: Aquest dispositiu va ser presentat a finals de 2015 per Sony, amb l'objectiu d'oferir un dispositiu de gama alta però de dimensions reduïdes. Amb una pantalla de 4.6", era gairebé l'únic dispositiu d'aquesta mida que oferia un processador Qualcomm Snapdragon 810 (la gama més alta de processadors que oferia Qualcomm en aquell moment). També cal destacar que aquest dispositiu suporta NFC i és resistent a l'aigua.

LeMobile Le X820: Aquest dispositiu també conegut com LeEco Le Max 2 va ser presentat el 2016 per LeEco. Aquest dispositiu es caracteritza per oferir característiques de gama alta a un preu molt menor. Amb un processador Snapdragon 820 i Android 6.0 de fàbrica, suposava una bona alternativa a altres dispositius de preus molt més alts.

5.2.2 Resultats obtinguts

Les mesures s'han realitzat per als tres tipus de servei (no reutilitzable, *m* vegades reutilitzable i infinitament reutilitzable). Per als serveis reutilitzables s'han realitzat les mesures dues vegades, una la primera vegada que el *user* emprava el servei i una altre vegada quan ja ha emprat el servei anteriorment. S'ha realitzat aquesta diferenciació perquè el procés de verificació és diferent per a la primera vegada que s'implementa el protocol què per a les vegades posteriors.

A la Figura 5.1 es pot observar el temps d'execució de la fase de verificació del PASS per a un servei no reutilitzable. Aquest temps d'execució es situa per davall dels 800ms per a tots els dispositius, fins i tot per als dispositius més antics.

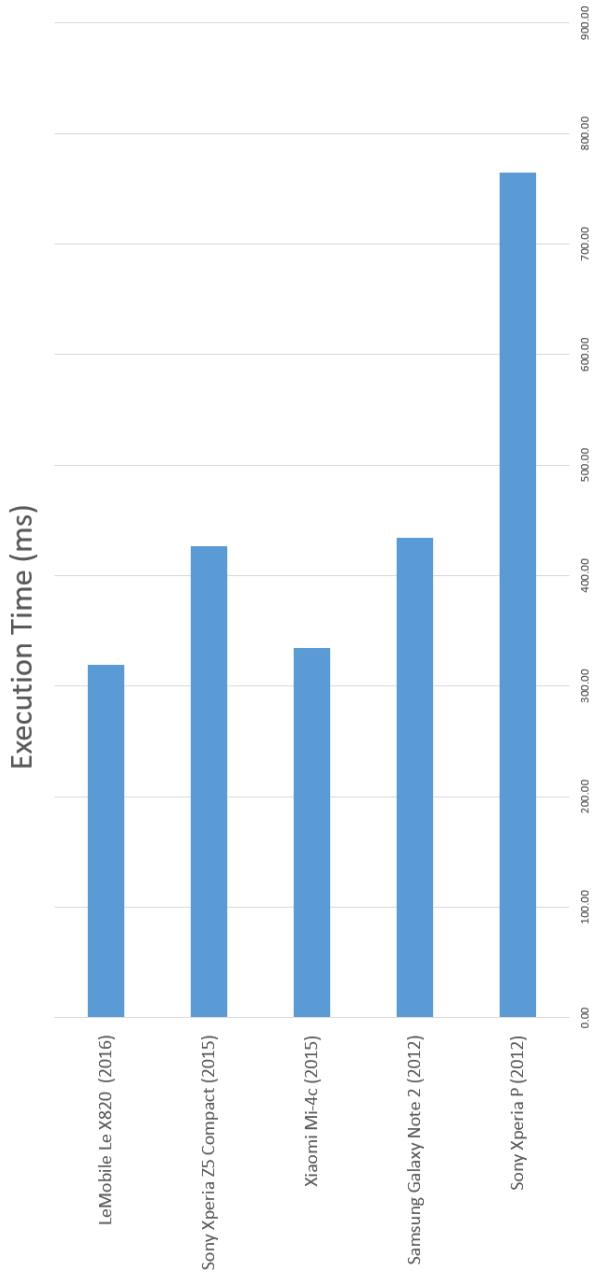

Figura 5.1: Mesura del temps d'execució de la fase de verificació del PASS per a un servei no reutilitzable

La Figura 5.2 mostra el temps d'execució de la fase de verificació del PASS per a un servei *m* vegades reutilitzable tan el cas de la primera de la verificació com de les posteriors. Tot i que el temps de verificació per a la primera vegada que s'empra el servei tendeix a ser major a quasi tots els dispositius, la diferència no suposa cap diferència notables ja que es situen en tots els casos per davall dels 400ms.

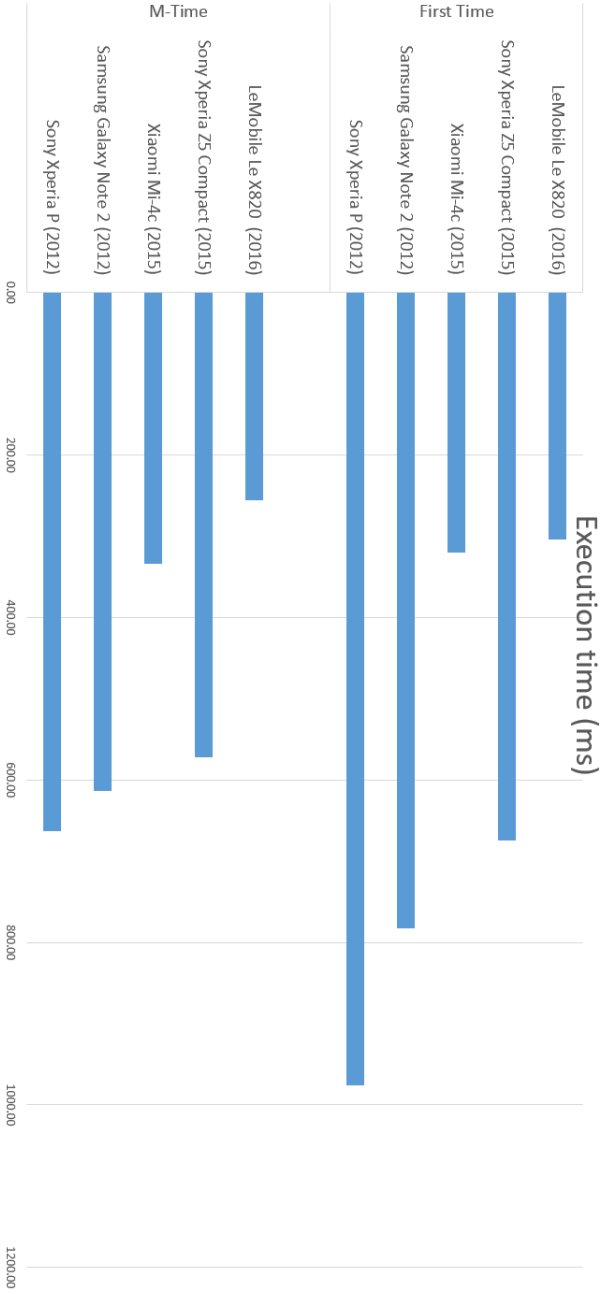

Figura 5.2: Mesura del temps d'execució de la fase de verificació del PASS per a un servei *m* vegades reutilitzable

La Figura 5.3 mostra el temps d'execució del PASS per a un servei infinitament reutilitzable. Com en el cas anterior, s'han realitzat les mesures tan per la primera vegada que s'empra el servei com per a les vegades posteriors. Aquestes mesures tornen a indicar que el temps d'execució del sistema és major per a la primera vegada. Aquest verificació és més ràpida que per a un servei *m* vegades reutilitzable ja que no s'ha de comprovar si s'ha intentat accedir-hi un determinat nombre de vegades.

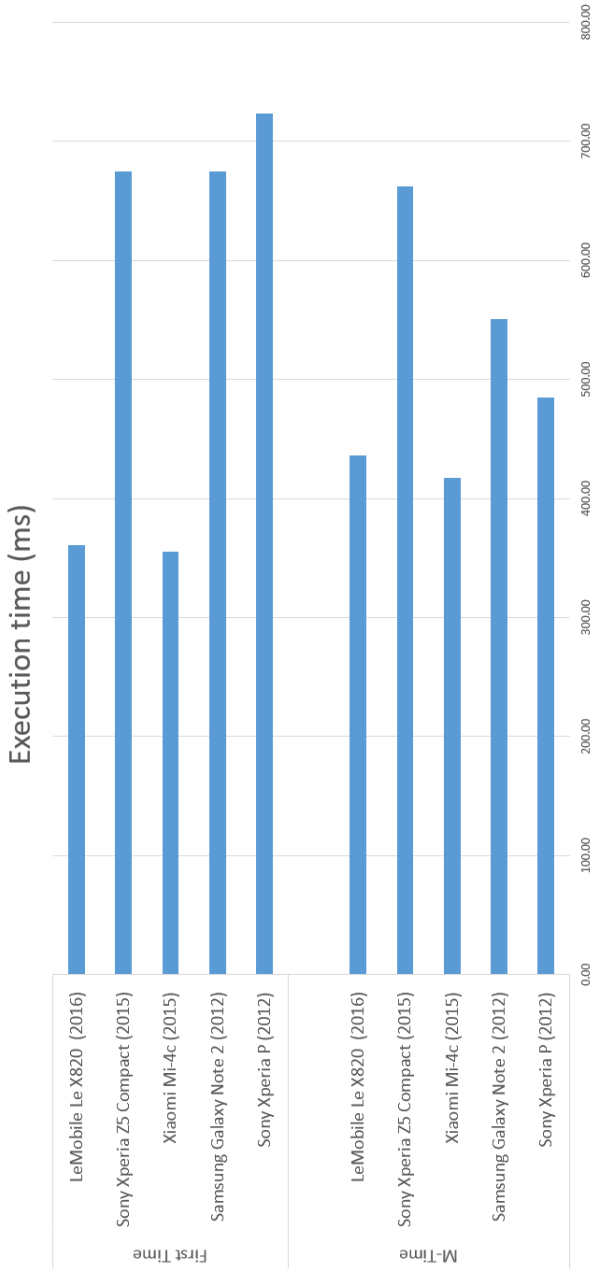

Figura 5.3: Mesura del temps d'execució de la fase de verificació del PASS per a un servei infinitament reutilitzable

5.2.3 Anàlisi dels resultats

Les mesures preses mostren que el temps d'execució de la fase de verificació es situa per davall del segon en tots els casos, fins i tot per a dispositius amb gairebé cinc anys d'antiguitat. Tot i que el temps d'execució és major la primera vegada que es verifica un *ticket* reutilitzable, la diferència es pot considerar gairebé negligible ja que entra dins un marge acceptable. Malgrat que les mesures s'han repetit diverses vegades, aquestes diferències podrien haver estat causades per l'execució d'altres processos en segon pla.

En conjunt, els resultats obtinguts mostren el potencial d'aquest protocol. Amb un temps d'execució menor a un segon, suposaria un gran oportunitat per a agilitzar les coes per a accedir tan a les atraccions com, sobretot, al transport públic.

5.3 Anàlisi del prototip

La tercera etapa de la implementació s'ha centrat en desenvolupar una aplicació que permetés al *user* realitzar totes les fases del protocol des del seu dispositiu mòbil.

Per a poder realitzar aquestes tasques, l'aplicació s'ha dividit en diferents pantalles amb les quals l'usuari podrà interactuar.

Generació d'un pseudònim L'aplicació no deixarà a l'usuari comprar un PASS fins que no hagi generat un pseudònim. A la pantalla de *Settings* mostrarà a l'usuari un botó que realitzarà aquest procés de manera transparent (Figura 5.4).

Figura 5.4: Pantalla per a generar un pseudònim

Una vegada ja hagi generat el pseudònim el contingut de la pantalla canviarà al mostrat a la Figura 5.5

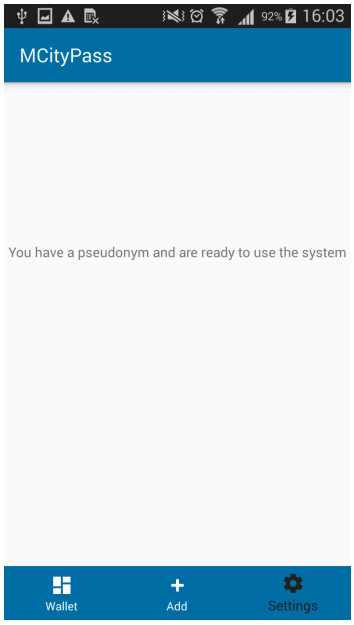

Figura 5.5: Pantalla amb el pseudònim generat

Comprar un PASS A la Figura 5.6 es mostra la pantalla amb la qual interactuarà l'usuari per a comprar un PASS. L'usuari podrà veure la data d'expiració (sis mesos després de la data de compra) i els serveis oferits dins aquell PASS i podrà elegir la categoria del PASS i el seu temps de vida.

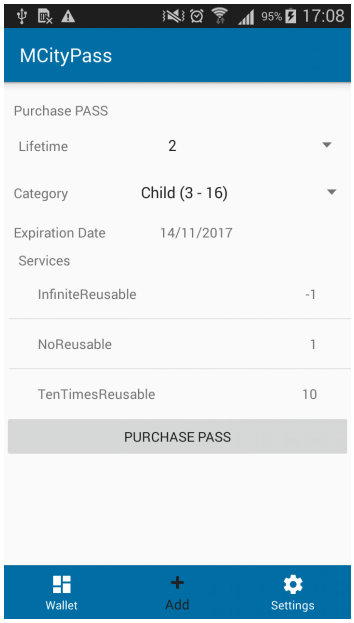

Figura 5.6: Interfície per a comprar un PASS

Llista dels PASS comprats A la Figura 5.7 es pot observar la pantalla de l'aplicació que mostra la llista dels PASS dels qual disposa el *user*.

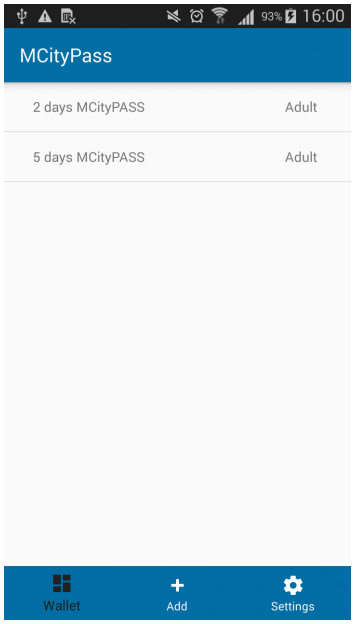

Figura 5.7: Llista dels PASS comprats

Informació d'un PASS Abans d'activar un PASS, l'usuari podrà veure la informació sobre aquest però no podrà fer ús de cap servei fins que no executi la fase d'activació d'un PASS (Figura 5.8).

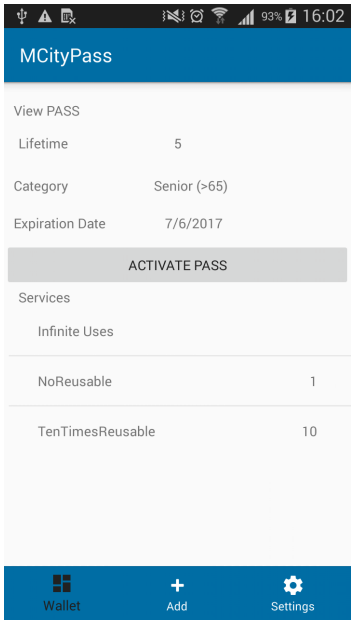

Figura 5.8: Informació sobre un PASS no activat

Per a fer ús d'un servei, l'usuari haurà d'elegir-l'ho de la llista i es començarà a executar el procés per a verificar un *ticket*. Si el procediment és satisfactori, indicarà a l'usuari que pot accedir a aquest servei (Figura 5.9). En el cas de què ja hagi fet ús

d'aquest servei el nombre definit de vegades, ja no li donarà opció d'emprar aquest *ticket* (Figura 5.10).

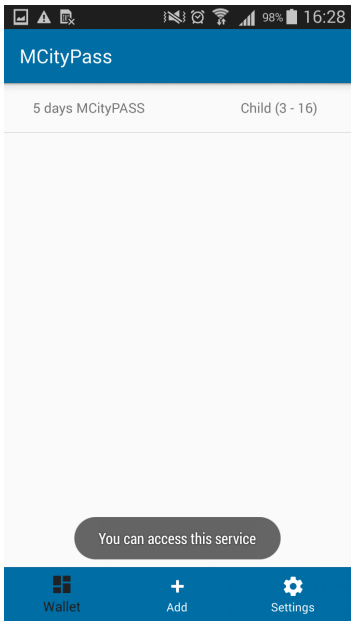

Figura 5.9: Accés a un servei

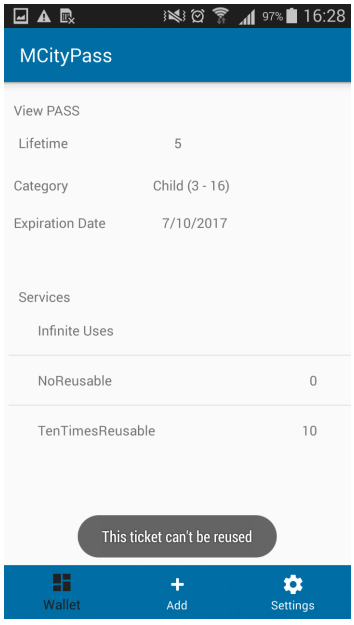

Figura 5.10: S'han esgotat els usos per a accedir a aquell servei

5.3.1 Limitacions del prototip

L'enfocament alhora de crear aquesta aplicació ha estat fer un prototip per a mostrar d'una manera aproximada les interaccions que tendria l'usuari amb el *framework* desenvolupat. Degut a aquest enfocament, el prototip té les següents limitacions:

- Cada usuari només pot tenir un pseudònim.
- Les connexions es realitzen amb un servidor connectat a la mateixa xarxa d'àrea local.
- Tots els PASS disposen dels mateixos serveis independentment de la seva durada.

CAPÍTOL 6

CONCLUSIONS

A mitjan segle XX, les millores en els mètodes de transport varen permetre a totes les persones visitar nous llocs per plaer. Això va suposar el naixement d'una nova industria basada en el turisme. Aquesta indústria avui en dia mou gran part de l'economia mundial, sent el principal motor de moltes regions.

Les millores tecnològiques dels darrers anys, amb el desenvolupament dels dispositius mòbils personals i mètodes de comunicacions cada vegada més ràpids han suposat una gran oportunitat per a reinventar aquesta industria. Aquestes tecnologies permeten oferir als usuaris serveis cada vegada més potents i personalitzats, alhora que acceleren i simplifiquen les interaccions necessàries.

Tot i així, moltes d'aquestes noves tecnologies es basen en la recopilació de dades de l'usuari, fent que aquest perdi el seu anonimat i es senti manco inclinat a fer ús d'aquestes.

En aquest document s'ha explicat una nova proposta creada per Payeras-Capellà et al. a l'article "*mCITYPASS : Privacy-preserving Secure Access to Federated Touristic Services with Mobile Devices*"[1]. Aquest article presenta un nou protocol orientat a garantir la seguretat i la privacitat de tot el sistema, així com l'anonimat dels usuaris honests. A més a més, es tracta d'un sistema flexible que permet l'entrada a serveis de diferents característiques i oferits per diferents proveïdors.

A continuació s'ha explicat el procediment per a realitzar un prototip d'aquest sistema. Per a realitzar aquesta tasca s'ha dividit el desenvolupament en diferents etapes, cada una orientada a resoldre un problema en particular de la implementació del sistema. A la primera s'ha garantit el correcte funcionament del protocol, observant si el procediment definit a l'article no presenta cap errada i és realment implementable.

La segona etapa s'ha orientat a obtenir valors aproximats del temps d'execució del procés de verificació per a accedir a un servei. Les mesures temporals s'han centrat en aquest moment del protocol ja que és el paràmetre més sensible alhora de dur el sistema a la realitat, ja que obtenir valors molt alts causaria coes a l'entrada dels serveis. La creació d'aquestes coes suposaria que aquesta proposta no és pot dur a la realitat, independentment de qualsevol altre de les qualitats del protocol.

Obtinguts resultats favorables, la darrera etapa s'ha centrat en desenvolupar un prototip que permetés a l'usuari accedir al *framework* i realitzar totes les fases definides en el protocol. Aquesta aplicació permet a l'usuari comprar PASS, mostra la informació sobre els PASS disponibles i permet fer-ne ús per a accedir a un servei.

6.1 Desenvolupament futur

A continuació es detallaran els possibles camins de desenvolupament que han quedat oberts.

- Modificació de les comunicacions per a emprar NFC. Aquesta modificació suposaria no només un canvi en les comunicacions sinó també que també s'hauria de modificar el protocol executat pels diferents actors. Tot i així seria un camí interessant per a millorar la usabilitat del sistema.
- Modificar l'aplicació per a ANDROID per a fer ús d'algun *framework* com IONIC per a desenvolupar una millor interfície gràfica.
- Creació d'una aplicació per a IOS. En el moment de realització d'aquest projecte, aquests dispositius no suporten el desenvolupament d'aplicacions que facin ús de les seves capacitats NFC i, per tant, s'hauria de decidir entre aquest camí i l'anterior.
- Desenvolupament d'un prototip més avançat amb més entitats i on cada actor s'executi en un entorn més semblant al qual s'executaria en una situació real.

BIBLIOGRAFIA

[1] M. Payeras-Capellà, J. Castell'a, M. Mut-Puigserver, and L. Huguet-Rotger, "mCITY-PASS : Privacy-preserving Secure Access to Federated Touristic Services with Mobile Devices," *Applied Computing and Information Technology, Springer's Studies in Computacional Intelligence*, To apperar in 2017. (document), 3, 3.1, 3.4, 3.5, 3.8, 3.9, 3.10, 3.11, 6

[2] M. Puhe, M. Edelmann, and M. Reichenbach, *Integrated urban e-ticketing for public transport and touristic sites*, 2014, no. January. 1, 1.1, 1.2, 1.3, 1.4

[3] A. Hanfried, B. Simon, B. Alexandre, B. Tony, C. Jean, F. Gino, H. Rainer, H. Johan, J. Helge, K. Steve, L. Olivier, L. Dietrich, M. Marcel, S. Sabine, T. Svend, E. Christian, and R. T. Ag, "Urban its expert group best practices in urban its," no. January, 2013. 1

[4] Over 65 million visits to london attractions in 2015. [Online]. Available: http://www.londonandpartners.com/media-centre/press-releases/2016/070316-over-65-million-visits-to-london-attractions-in-2015 1.1

[5] K. Mori and A. Shiibashi, "Trend of autonomous decentralized system technologies and their application in ic card ticket system," *IEICE Transactions on Communications*, vol. E92-B, no. 2, pp. 445–460, 2009. 2.1.1

[6] R. Couto, J. Leal, P. M. Costa, and T. Galvao, "Exploring Ticketing Approaches Using Mobile Technologies: QR Codes, NFC and BLE," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2015-Octob, pp. 7–12, 2015. 2.1.2

[7] M. Mut-Puigserver, M. M. Payeras-Capellà, J. L. Ferrer-Gomila, A. Vives-Guasch, and J. Castellà-Roca, "A survey of electronic ticketing applied to transport," *Computers and Security*, vol. 31, no. 8, pp. 925–939, 2012. 2.1.3, 2.1.3, 3.1

[8] H. Rodrigues, R. José, A. Coelho, A. Melro, M. C. Ferreira, J. F. e Cunha, M. P. Monteiro, and C. Ribeiro, "Mobipag: Integrated mobile payment, ticketing and couponing solution based on NFC," *Sensors (Switzerland)*, vol. 14, no. 8, pp. 13 389–13 415, 2014. 2.2.1

[9] Emvco. [Online]. Available: https://www.emvco.com/specifications.aspx?id=21 2.2.1

[10] A. Basili, W. Liguori, and F. Palumbo, "NFC smart tourist card: Combining mobile and contactless technologies towards a smart tourist experience," *Proceedings of*

the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE, pp. 249–254, 2014. 2.2.2

[11] M. C. Ferreira, H. Nóvoa, T. G. Dias, and J. F. e. Cunha, "A Proposal for a Public Transport Ticketing Solution based on Customers' Mobile Devices," *Procedia - Social and Behavioral Sciences*, vol. 111, pp. 232–241, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.sbspro.2014.01.056 2.2.3

[12] D. Antunes, J. Lima, G. Pereira, N. Escravana, and C. Ribeiro, "NFC4Sure: Mobile ticketing system," *Proceedings of the 2016 2nd Conference on Mobile and Secure Services, MOBISECSERV 2016*, 2016. 2.2.4

[13] Sql vs nosql: How to choose. [Online]. Available: https://www.sitepoint.com/sql-vs-nosql-choose/ 4.1

[14] Gartner says worldwide sales of smartphones grew 7 percent in the fourth quarter of 2016. [Online]. Available: http://www.gartner.com/newsroom/id/3609817 4.1, 4.5.2

[15] X.509. [Online]. Available: https://en.wikipedia.org/wiki/X.509 4.1.1

[16] Spring. [Online]. Available: https://projects.spring.io/spring-framework/ 4.4.2

[17] What are restful web services? [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html 4.4.2

[18] Springboot. [Online]. Available: https://projects.spring.io/spring-boot/ 4.4.2

[19] Gradle. [Online]. Available: http://www.drdobbs.com/jvm/why-build-your-java-projects-with-gradle/240168608 4.4.2

[20] Class biginteger. [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html 4.4.2

[21] Hsqldb. [Online]. Available: http://hsqldb.org/ 4.4.2

[22] Hibernate orm. [Online]. Available: http://hibernate.org/orm/ 4.4.2

[23] Thymeleaf. [Online]. Available: http://www.thymeleaf.org/ 4.4.2

[24] Model view controller. [Online]. Available: https://goo.gl/8KCZuq 4.4.2

[25] Lean service architectures with java ee 6. [Online]. Available: http://www.javaworld.com/article/2078031/java-se/lean-service-architectures-with-java-ee-6.html 4.4.3

[26] Data access objects. [Online]. Available: https://en.wikipedia.org/wiki/Data_access_object 4.4.3

[27] Ventas de smartphones: Android se come a windows. [Online]. Available: http://goo.gl/zM2gy4 4.5.2

[28] Retrofit. [Online]. Available: http://square.github.io/retrofit/ 4.5.2

[29] Hard coding. [Online]. Available: https://en.wikipedia.org/wiki/Hard_coding 4.5.3

[30] About sqlite. [Online]. Available: https://www.sqlite.org/about.html 4.6.2

[31] Most widely deployed and used database engine. [Online]. Available: https://www.sqlite.org/mostdeployed.html 4.6.2

[32] Activeandroid. [Online]. Available: http://www.activeandroid.com/ 4.6.2

[33] Application fundamentals. [Online]. Available: https://developer.android.com/guide/components/fundamentals.html 4.6.3