mCITYPASS: Privacy-preserving Secure Access to Federated Touristic Services with Mobile Devices

Macià Mut-Puigserver *, M. Magdalena Payeras-Capellà *, Jordi Castellà-Roca †, Llorenç Huguet-Rotger *
* Dpt. de C. Matemàtiques i Informàtica, Universitat de les Illes Balears,
Ctra. de Valldemossa, km 7,5. E-07122 Palma, Spain. Email: {macia.mut, mpayeras,l.huguet}@uib.cat
† Dpt. d'Enginyeria Informàtica i Matemàtiques, UNESCO Chair in Data Privacy, Universitat Rovira i Virgili.
Av. Països Catalans 26, E-43007 Tarragona, Spain. Email: jordi.castella@urv.cat

***Abstract*—Destination cards are offered in many cities to provide tourists a simple way to access interest points and public transport. Existing e-ticketing proposals are general purpose systems or applied to transport services. In this paper we present an electronic ticketing system intended to be used for touristic services. Usually these cities have inneficient CITYPASS systems implemented on smart cards. Our proposal is the first one that can be implemented on portable devices, such as smartphones, and is flexible enough to include reusable and non-reusable services in the same PASS. The design of the system has been designed taking into account all the security and privacy requirements described for electronic tickets, including the challenging ones (exculpability, reusability and unsplittability) resulting a very secure and powerful system. The dispute resolution protocol assures that all the parts are protected against other part's attacks. Finally, the system allows the user to use the system anonymously, so the privacy of the system is assured.**

Keywords: secure access, tourism, e-ticketing, e-commerce, security, privacy, exculpability, smart cities.

I. Introduction

Tourism is subject to a transformation due to the use of communication technology. ICT is an enabling and integrative technology in nearly all areas of daily life, and it also affects urban transport and tourism. Almost all big cities or touristic areas offer an integrated service to access the touristic sites (museums, monuments, attractions, exhibitions,...) and the public transport. These integrated ticketing schemes are called destination cards or city passes and are valid for an specific period (lifetime). Along with tickets, the schemes habitually offer direct access skipping queues and providing discounts in complementary services [21]. Usually the destination cards are implemented over smart cards (activated before its first use), see figure 1, or paper booklets including bar codes. Examples of these cards can be found in "We love city cards" [6], a portal to destination card systems of 36 european cities or CityPASS [10] a portal to 12 american cities. The most relevant advantages of the offers are free entry to top visitor attractions, offers and discounts, priority access, choice of cards for different periods, free guidebook, city maps and limited or unlimited access to public transport. We propose a new model for destination cards based on the use of portable devices, such as smartphones. mCITYPASS is an application that allows the purchase and use of a destination card that includes multiple access tickets. According to the services these tickets can be non-reusable, m-times reusable or infinitely reusable. In real scenarios, non-reusable tickets are usually related with attractions while reusable tickets are usually related with transport, but the specific use depends on the city. As an example, Granada CityPASS (lovegranada.com) offers a three day pass that includes The Alhambra and Generalife entrance and 5 urban bus journeys while the pass for five days includes The Alhambra and Generalife entrance, 9 urban bus journeys and 1 tourist train trip. Together with reusability the system must protect the privacy of the user to avoid the generation of identified user profiles and must be secure enough to avoid forgery. Finally, the use of the system must represent a fair scenario for both users and service providers. The proposed system fulfills the security and privacy requirements for a destination pass system based on mobile devices.

Fig. 1. Example of Destination Cards. Font: welovecards.com

A. Organization

This paper is organized as follows. Section II reviews the state of the art, including security requirements and related proposals. Section III describes in detail all the phases of the protocol while IV includes the claims that guarantee the fairness and the exculpability of the scheme. Section V includes a brief security and privacy analysis and finally VI list the conclusions of the paper.

II. State of the art

In this section we will describe the state of the art in destination cards. First, the security requirements for this kind of applications are listed and then the most significant proposals are described and evaluated.

A. Security requirements

The destination card systems have to consider and guarantee the following security requirements, described in [18]:

- Authenticity: A user has to be able to verify if a PASS has been issued by an authorized issuer.
- Integrity: All the parts have to be able to verify if the PASS has been altered as regards the one issued by the correspondent authorized issuer.
- Non-repudiation: Once a valid PASS has been issued, the issuer has not to be able to deny that he has issued that PASS.
- Unforgeability: Only authorized issuers can issue a valid PASS.
- Non-Overspending and reusability: The PASS will include several tickets, both non-reusable and reusable. In both cases, ticket overspending has to be prevented. Tickets included in the PASS can only be used as agreed between the issuer and the user. Non-reusable tickets can not be reused after they have been spent. Reusable tickets can be used exactly the number of times agreed in the moment of issue.
- Revocable anonymity: The scheme allows the revocation of the anonymity of the user if he misbehaves using the service, otherwise the user remains anonymous.
- Expiration date and lifetime: A PASS will be only valid during a time interval.
- Fairness: During the execution of the protocol the parties execute several exchanges of elements, this exchanges could not lead to unfair situations.
- Exculpability: The provider can not falsely accuse the user of ticket overspending, and the user is able to demonstrate that he has already validated the ticket before using the service.
- Unsplittability: Users should not be able to share a PASS in such a way that several users validate tickets from the same PASS.

B. Related proposals

There are some papers that explore the idea of the development of integrated e-ticketing systems for touristic sites in cities. The idea behind these tickets is to combine several touristic services (e.g. transport passes, leisure activities tickets, tourist attractions vouchers) on a single ticket. The integrated ticket has the advantage of having a lower fare than buying the passes of the different services separately. For example, these systems became a goal of the European Union applied to the transports systems in cities [23]. Though, e-ticketing systems must be secure and have to introduce measures to avoid counterfeit copies and preserve user's privacy.

Many large cities have introduced multimodal e-ticket systems in public transportation and tourist attractions [6], [10]. The technology used in them may vary from one city to another (e.g. contactless or contact-based smartcards). Nevertheless, the use of smartphones remains residual. In Helsinki, Finland, public transport users have been able to pay their fares with their mobile phones since 2001 [17]. However, in some cases (e.g. New York CityPASS), vouchers must be printed and exchanged for the user's CityPASS booklet at the visited attraction. In [5] is explained the practical advantages of operating with mobile devices, which reduces the cost of system infrastructure.

Also, there some are papers [14], [9], [1], [24] that explore the advantages and the development of e-ticketing schemes in touristic environments and transport systems. Nevertheless, just a few of them have some specification on how to implement such systems. Alessadra et al. in [4] describes an application for Android smartphones offering, among others, mobile ticketing services. However, the paper only presents a generic framework and does not describe any e-ticketing implementation.

In some papers mobile phones are used as virtual vouchers, e-tickets applied to the transport system, or even supermarket loyalty cards. Sang-Won et al. in [3] suggests a NFC-based mobile ticket for small traders and enterprisers that can be used in touristic applications. However, this proposal does not store the ticket on the smartphone, they just use the mobile device to identify the user, thus anonymity is not possible in such schemes. In [12] the author proposes the so called $mCoupons$ stored in the mobile device. The scheme proposed in this paper is only applied to single tickets and the security of the systems only deals with against multiple spending, unauthorized generation and manipulation, and copying. Han-Cheng in [13] proposes a similar scheme (same features than [12] but a more efficient implementation).

Multi-ticketing systems were introduced to increase the efficiency of having tickets separately [16]. Chen et al. [7] introduced a multi-ticketing system, which provides privacy-protection and also protection against splitting. However, the tickets contained in the same multi-ticket can only be redeemed in the order that was fixed during the issue protocol. Some inefficient issues of the Chen's protocol were improved in [16]. In any case, only a single vendor is considered.

In order to make more appealing to users, multi-ticketing systems with a federation of vendors has been designed in [2]. In this case, the mobile device can store a multi-ticket token that contains single-tickets from a federation of vendors (e.g. a corporation of touristic attractions managed by different vendors from the same city). This proposal presents unsplittability but reusability, exculpability, activation and fairness are not contemplated in the system.

III. mCITYPASS scheme

The scheme has the following actors: the user $\mathcal{U}$; the ticket issuer $\mathcal{I}$, who sends a valid ticket to $\mathcal{U}$; a set of service providers P_i, who verify the tickets inside the PASS

and give the corresponding service; and finally a trusted third party (TTP) $\mathcal{T}$, who preserves $\mathcal{U}$'s anonymity, and gives a valid non-identity pseudonym to $\mathcal{U}$. The mCITYPASS (mobile CITYPASS) scheme has been designed for mobile devices (smartphones, tablets, smartwatches,..), reducing the computation requirements in the user side, and providing the basic security requirements (authenticity, non-repudiation and integrity) together with expiry and activation date, revocable anonymity, exculpability, reusability and unsplittability.

In Table I and Table II we define the notation used in the description of our scheme.

A. Phases

The phases of our system are: *Providers' Affiliation*, that includes the procedure to affiliate each Service Provider and the procedure to generate the parameters of the mCITYPASS scheme; *Pseudonym*, where the user obtains a new temporary pseudonym to be used in the system without linkage to user's identity (if user behaves correctly); *PASS Purchase*, that consists on the payment and reception of the PASS; the *PASS Activation* that can be executed together with the *PASS Purchase* o later, and *PASS Verification*, where the user shows a ticket included in the PASS to a service provider in order to be checked and validated. Other phases considered in the system are claims, that should only be executed in case of controversial situations during the *PASS Verification* phase.

Users have a digital credential ($\mathsf{Cert}_{\mathcal{U}}$) only for authentication to the TTP, since the system is anonymous, and all further movements in the system are tracked only with the assigned temporary pseudonym ($\mathsf{Pseu}_{\mathcal{U}}$).

1) PHASE 1: Providers' Affiliation: The first stage of the scheme is the affiliation of the service providers. They have to contact with the issuer $\mathcal{I}$ and enroll as a provider of a certain service in the framework of the mCITYPASS. The number of services included in the PASS are denoted by J, so J providers are going to be joined to the scheme. As a result, $\mathcal{I}$ will have an array where all public key certificates of the providers are arranged. A provider P_i can supply either a non-reusable service (γ_i), or a m-times reusable service (λ_i) (this means that each reusable service λ_i has a maximum number of uses identified by m) or a infinitely reusable service (ξ_i). The protocol is as follows:

$\mathcal{I}$ 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 $\mathcal{I}$ via an authenticated channel;

generateCertificate

Issuer $\mathcal{I}$ 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);

When all J providers have executed the previous protocol the issuer can generate the array of certificates:

Fig. 2. Example of the seeds included in a PASS with 7 services: one totally reusable service, three non-reusable services, two four-time reusable services and one twice-reusable service

arrayGeneration

Issuer $\mathcal{I}$ executes:

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

After the affiliation of the P_i, the structure of the PASS can be created (see figure 2).

2) PHASE 2: Pseudonym: The user $\mathcal{U}$ contacts the pseudonym manager $\mathcal{T}$ in order to obtain the assigned pseudonym. The certificate $\mathsf{Cert}_{\mathcal{U}}$ identifies $\mathcal{U}$ through a secure connection established between the two parties. The system has cryptographic public parameters [22], which have been published: (α, p, q), where α is a generator of the group G with order p, being p and q large primes achieving $p = 2q+1$. $\mathcal{U}$ generates a random value $\mathsf{x}_{\mathcal{U}} \overset{R}{\leftarrow} \mathbb{Z}_q$ and computes $\mathsf{y}_{\mathcal{U}} = \alpha^{\mathsf{x}_{\mathcal{U}}} \pmod p$ in order to receive a valid signed pseudonym $\mathsf{Pseu}_{\mathcal{U}}$ from $\mathcal{T}$. $\mathcal{U}$ and $\mathcal{T}$ have their own pair of keys used for signature and encryption of the transmitted data between them. The protocol is as follows:

authenticateUser

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

1) generates $\mathsf{x}_{\mathcal{U}} \overset{R}{\leftarrow} \mathbb{Z}_q$, and computes $\mathsf{y}_{\mathcal{U}} = \alpha^{\mathsf{x}_{\mathcal{U}}} \pmod p$;
2) computes the signature $\mathsf{sk}_{\mathcal{U}}(\mathsf{h}_{\mathsf{y}_{\mathcal{U}}})$ where $\mathsf{h}_{\mathsf{y}_{\mathcal{U}}} = hash(\mathsf{y}_{\mathcal{U}})$;
3) encrypts $\mathsf{y}_{\mathcal{U}}$ with the $\mathcal{T}$'s public key: $\mathsf{pk}_{\mathcal{T}}(\mathsf{y}_{\mathcal{U}})$;
4) sends $(\mathsf{sk}_{\mathcal{U}}(\mathsf{h}_{\mathsf{y}_{\mathcal{U}}}), \mathsf{Cert}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{T}}(\mathsf{y}_{\mathcal{U}}))$ to $\mathcal{T}$;

generatePseudonym

Pseudonym Manager $\mathcal{T}$ executes:

1) decrypts $\mathsf{sk}_{\mathcal{T}}(\mathsf{pk}_{\mathcal{T}}(\mathsf{y}_{\mathcal{U}})) \rightarrow (\mathsf{y}_{\mathcal{U}})$;
2) verifies $\mathsf{y}_{\mathcal{U}}$: $\mathsf{pk}_{\mathcal{U}}(\mathsf{sk}_{\mathcal{U}}(\mathsf{h}_{\mathsf{y}_{\mathcal{U}}})) \rightarrow (\mathsf{h}_{\mathsf{y}_{\mathcal{U}}}) \overset{?}{=} hash(\mathsf{y}_{\mathcal{U}})$;
3) if correct, then computes the signature of $\mathsf{sk}_{\mathcal{T}}(\mathsf{h}_{\mathsf{y}_{\mathcal{U}}})$; and
4) sends $\mathsf{Pseu}_{\mathcal{U}} = (\mathsf{y}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{T}}(\mathsf{h}_{\mathsf{y}_{\mathcal{U}}}))$ to $\mathcal{U}$.

verifyPseudonym

$\mathcal{U}$ computes:

1) verifies $\mathsf{y}_{\mathcal{U}}$: $\mathsf{pk}_{\mathcal{T}}(\mathsf{sk}_{\mathcal{T}}(\mathsf{h}_{\mathsf{y}_{\mathcal{U}}})) \rightarrow (\mathsf{h}_{\mathsf{y}_{\mathcal{U}}}) \overset{?}{=} hash(\mathsf{y}_{\mathcal{U}})$;

If in the future the user visit the same city and wants to buy another PASS, he can execute this phase again in order to avoid *vinculability* in the usage of the anonymous tickets.

3) PHASE 3: PASS Purchase: The user establishes a connection with the PASS issuer $\mathcal{I}$ in order to receive it. This connection could be established through an anonymous channel like TOR [11], guaranteeing then user's privacy. There are current contributions [1] that have implemented TOR for Android devices. $\mathcal{I}$ has a key pair and a public key certificate ($\mathsf{Cert}_{\mathcal{I}}$). Users do not use their personal keys (it would cause loss of anonymity); instead they use the temporal pseudonym and authenticate through the Schnorr's Zero-Knowledge Proof (ZKP) [20]. The payment method is considered as out of scope in this proposal as we focus on the privacy given to user when joining the system, and using the services.

[1] http://sourceforge.net/apps/trac/silvertunnel/wiki/TorJavaOverview

TABLE I
CRYPTOGRAPHIC PRIMITIVES USED IN THE PROTOCOL DESCRIPTION

CRYPTOGRAPHY: Notation and Description			
$\mathsf{sk}_{\mathcal{E}}(content)$	Decryption of $content$ or the generation of a signature with its $content$ by using the private key of the entity $\mathcal{E}$	$\mathsf{pk}_{\mathcal{E}}(content)$	Encryption of $content$ or the verification of a signature $content$ by using the public key of the entity $\mathcal{E}$
$hash()$	Public cryptographic one-way summarizing function that achieves collision-resistance	$hash^m()$	Represents the hash function applied m times

TABLE II
DATA STRUCTURES AND ITEMS USED IN THE PROTOCOL DESCRIPTION

INFORMATION ITEMS: Notation and Description			
$\mathsf{Cert}_{\mathcal{U}}$	User's Digital Credential	$\mathsf{Pseu}_{\mathcal{U}}$	User Temporary pseudonym
$Cert_{\mathcal{P}}[1..J]$	Array of all the providers' certifications	Sn	Serial number of the PASS
γ_i	Index for a non-reusable service	λ_i	Index for a reusable service
ξ_i	Index for an infinitely reusable service	K	Shared session key
RI_i	Secret random value sent by P_i to $\mathcal{U}$ in order to give her the right to use a service	RU	Secret random number to demonstrate the ownership of a PASS
$\delta_{\mathcal{T},P_i}$	Digital envelope of κ_i	$\delta_{\mathcal{T},\mathcal{P}[\,]}$	Array of all digital envelopes for all services in a PASS
$\mathsf{h}_{\mathsf{RI}[\,]}$	Array with all hashes on all sercret radom numbers RI_i	h_{RU}	Result of the hash function applied to RU
τ_1	First verfication timestamp. P_i generates it during the verification process of a PASS at *verifyPASS* stage	τ_2	Second verfication timestamp. P_i creates it during the verification process of a PASS at *verifyProof* stage
$\mathsf{A}_{\mathcal{P}}$	Vernam cipher of RI_i	$\mathsf{A}_{\mathcal{U}}$	Vernam cipher of $\psi_{i,j}$
k_{λ_i}	Counter to keep track of the number of times that the ticket can be used	ACT	Activation proof of a PASS
$\psi_{i,0}$	Secret value, the knowlege of this value by $\mathcal{U}$ proves the right to use a non-reusable service	$\psi_{i,j}$	Value derived from $\psi_{i,0}$ using j-times a hash function. The knowlege of this value by $\mathcal{U}$ proves the right to use a reusable service
$\mathsf{V}_{\mathsf{succ}}$	Message generated by P_i, the meaning of this message is denoted by flag_1 or flag_2	$\mathsf{V}_{\mathsf{fail}}$	Message generated by P_i, the meaning of this message is denoted by flag_0
κ_i	Information generated by $\mathcal{I}$ to P_i and $\mathcal{T}$ to give $\mathcal{U}$ the right to use a service	flag_{10}	Indicates that the ZKP has finished successfully. So, the service can still be used by $\mathcal{U}$ using the PASS. In case of a reusable service, it also specify that P_i agrees with counter k_{λ_i}
flag_{00}	It is used by P_i to indicates whether any parameter of the PASS sent by $\mathcal{U}$ has not the proper form (e.g. the current date is past the final expiry date) or its signature are not correct	flag_{01}	Indicates the problem: ZKP ends unsuccessfully ($\mathcal{U}$ has not demonstrate the ownership of the PASS)
flag_{02}	Indicates that the ticket has been spent, i.e. is not valid anymore	flag_{03}	Specifies two possible errors: $\psi_{\gamma_i,0}$ does not match with $\psi_{\gamma_i,1}$, or τ_2 does not match the expiry date of the PASS
flag_{04}	Indicates that the ticket is not valid or some of the parameters sent by $\mathcal{U}$ are not correct	flag_{05}	It specifies that $\psi_{\lambda,(k_{\lambda_i}-1)}$ received in the previous step is not correct or τ_2 does not match the expiry date of the PASS

In this phase, the user selects a PASS Type, that defines the duration of the PASS, Lifetime, and the Category of the user (adult, child, young,..): Type(Lifetime, Category). $\mathcal{I}$ generates the PASS with all the required information and its digital signature, together with an array of J secret values RI_i (where $i = 1..J$, let J be the number of providers affiliated to the PASS service) and the set of secret shared keys (they are decryptable only by the corresponding provider P_i and $\mathcal{T}$) in order to let each provider show the secret value RI_i related to each provider later, in the verification phase. The ticket issuer $\mathcal{I}$ and the user $\mathcal{U}$ follow this protocol:

getService

$\mathcal{U}$ 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 $\mathsf{h}_{\mathsf{RU}} = hash(\mathsf{RU})$;
3) computes $\mathsf{H}_{\mathcal{U}} = \alpha^{\mathsf{RU}} \pmod p$;
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 $(\mathsf{Pseu}_{\mathcal{U}}, \mathsf{H}_{\mathcal{U}}, A_1, A_2, \mathsf{h}_{\mathsf{RU}}, \mathsf{Type})$ to the ticket issuer $\mathcal{I}$.

getChallenge

$\mathcal{I}$ follows the next steps:

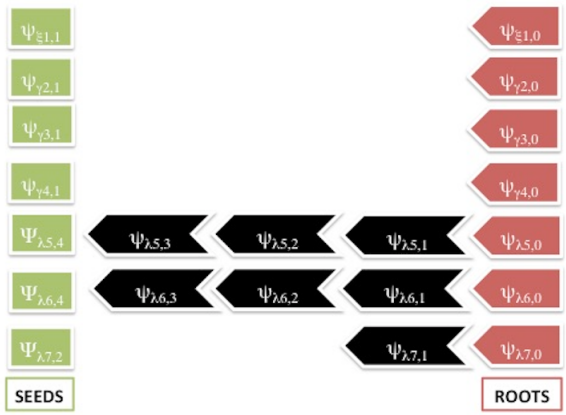
1) generates and sends a challenge $\mathsf{c} \xleftarrow{R} \mathbb{Z}_q$ for $\mathcal{U}$;
2) asynchronously, for optimization, pre-computes $\mathsf{y}_{\mathcal{U}}{}^{\mathsf{c}} \pmod p$ and $\mathsf{H}_{\mathcal{U}}{}^{\mathsf{c}} \pmod p$;

solveChallenge

$\mathcal{U}$ computes:

1) computes $\mathsf{w}_1 = a_1 + \mathsf{c} \cdot \mathsf{x}_{\mathcal{U}} \pmod q$;
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 $hash^m()$ represents the hash function applied m times; see figure 3
7) stores $\psi_{1,0}, ..., \psi_{J,0}$ values in her *TicketsPASS* database;
8) encrypts and sends the generated information to $\mathcal{I}$: $\mathsf{pk}_{\mathcal{I}}((\mathsf{w}_1, \mathsf{w}_2), (\{\psi_{\gamma_i,1}, \psi_{\lambda_i,m}\}\forall\gamma_i, \lambda_i))$ and pays for the PASS;

Fig. 3. Generation of the seed for an Example of a PASS with 7 services: one totally reusable service, three non-reusable services, two four-time reusable services and one twice-reusable service



getPASS

$\mathcal{I}$ follows the next steps:

1) decrypts $\mathsf{sk}_{\mathcal{I}}(\mathsf{pk}_{\mathcal{I}}(\mathsf{w}_1, \mathsf{w}_2)) \rightarrow (\mathsf{w}_1, \mathsf{w}_2)$;
2) computes $\alpha^{\mathsf{w}_1} \pmod p$;

3) computes $\alpha^{\mathsf{w}_2} \pmod p$;
4) verifies $\alpha^{\mathsf{w}_1} \stackrel{?}{=} A_1 \cdot \mathsf{y}_{\mathcal{U}}{}^{\mathsf{c}} \pmod p$;
5) verifies $\alpha^{\mathsf{w}_2} \stackrel{?}{=} A_2 \cdot \mathsf{H}_{\mathcal{U}}{}^{\mathsf{c}} \pmod p$;
6) if both verifications hold, $\mathcal{I}$ has checked that $\mathcal{U}$ is an authorized user. Otherwise, $\mathcal{I}$ 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 $\mathsf{RI}_i \stackrel{R}{\leftarrow} \mathbb{Z}_p$, for $i = 0, , J$;
9) computes the set $\mathsf{h}_{\mathsf{RI}_i} = hash(\mathsf{RI}_i)$ for $i = 1, , J$;
10) composes the set $\kappa_i = (\mathsf{K}, \mathsf{RI}_i)$ and signs it $\kappa_i^* = (\kappa_i, \mathsf{Sign}_{\mathcal{I}}(\kappa_i)) \forall i = 1, ..., J$;
11) encrypts each κ_i^* with a digital envelope which is decryptable by the TTP $\mathcal{T}$ and the provider P_i for possible future controversial situations during the ticket verification: $\delta_{\mathcal{T},P_i} = \mathsf{pk}_{\mathcal{T},\mathcal{P}_i}(\kappa_i^*)$.
12) fills out the PASS information PASS = (Sn, Type, $\mathsf{Pseu}_{\mathcal{U}}$, Lifetime, PURdate, EXPdate, $\mathsf{h}_{\mathsf{RI}}[\ ]$, h_{RU}, $\delta_{\mathcal{T},\mathcal{P}}[\ ]$, $\psi_{\lambda,m}[\ ]$, $\psi_{\gamma,1}[\ ]$, Terms and Conditions);
13) digitally signs the PASS, $\mathsf{Sign}_{\mathcal{I}}(\mathsf{PASS}) = \mathsf{sk}_{\mathcal{I}}(hash(\mathsf{PASS}))$, and generates $\mathsf{PASS}^* = (\mathsf{PASS}, \mathsf{Sign}_{\mathcal{I}}(\mathsf{PASS}))$;
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 $\mathcal{U}$.

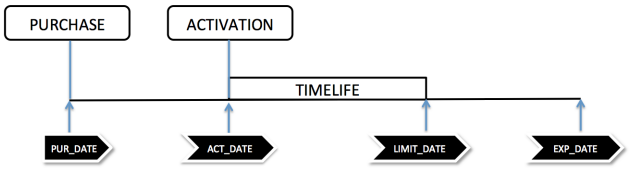
receivePASS

$\mathcal{U}$ executes:

1) verifies the digital signature $\mathsf{Sign}_{\mathcal{I}}(\mathsf{PASS})$ 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 $\mathsf{PASS.Pseu}_{\mathcal{U}}$;
5) stores in her *TicketsPASS* database $(\mathsf{PASS}^*, \mathsf{RU})$ together with the associated information stored in the step **solveChallenge**.7 of this phase.

4) PHASE 4: PASS Activation: One important difference between standard tickets and a CITYPASS is that whereas tickets can be used anytime before its expiration date, a PASS has a Lifetime together with its expiration date, EXPdate. That means that the PASS must be used before its expiration date and will be valid during a Lifetime period (defined during the Pass purchase, Type(Lifetime, Category)). This Lifetime period begins when the user activates the PASS. The activation can be performed immediately after the purchase or it can be performed just before the first use of the PASS.
We have described an activation between $\mathcal{U}$ and $\mathcal{I}$ but it can be performed between $\mathcal{U}$ and P_i.

Fig. 4. PASS' Life Cycle



Each PASS has a Purchase date, PURdate and an expiration date EXPdate fixed during the Pass Purcahse phase. Moreover, each user selects the type of card he wants to buy, including the validity period of the PASS, Lifetime. The activation phase sets the value of the activation date ACTdate. The the limit date LIMdate is calculated adding the Lifetime to ACTdate. After LIMdate the services of the card cannot be used anymore. If the card is not activated before EXPdate or if LIMdate is greater than EXPdate, then the services cannot be used after EXPdate (see figure 4). The protocol is as follows:

showPass

$\mathcal{U}$ computes:

1) sends PASS^* to $\mathcal{I}$;

verifyTicket

$\mathcal{I}$ executes:

1) verifies the PASS signature, PASS.PURdate, and PASS.EXPdate;
2) if the verifications fail, $\mathcal{I}$ aborts the ticket activation;
3) else $\mathcal{I}$ looks for the PASS PASS^* in the database using PASS.Sn; and
verifies that the PASS has not been activated: $\stackrel{?}{\exists}$ ACT linked to PASS^* in the DB;
   a) if $\nexists$ACT linked to PASS^*:
      i) generates $\mathsf{ACT} = (\mathsf{PASS.Sn}, \mathsf{ACTdate}, \text{"}Activated\text{"})$ and digitally signs ACT, and obtains the signed activation proof, $\mathsf{Sign}_{\mathcal{I}}(\mathsf{ACT}) = \mathsf{sk}_{\mathcal{I}}(hash(\mathsf{ACT}))$, and $\mathsf{ACT}^* = (\mathsf{ACT}, \mathsf{Sign}_{\mathcal{I}}(\mathsf{ACT}))$;
      ii) sends ACT^* to the user $\mathcal{U}$.

5) PHASE 5: PASS Verification: When the user wants to access the service, he must show and verify the PASS in advance. The user only interacts with the service provider, but in controversial situations, he and/or the service provider could interact directly with the TTP through a *resilient channel*[2] in order to preserve the security requirements of the protocol. If the user misbehaved, her identity could be revoked, enabling to take further actions.
$\mathcal{U}$ sends the PASS^*, and $\mathcal{P}$ checks it. If passed, $\mathcal{P}$ sends the commitment so that the corresponding RI will be disclosed if $\mathcal{U}$ behaves correctly. Once the user proves the knowledge of the value RU (related to the PASS) using a ZKP and reveals the secret value corresponding to the service, then he receives the secret RI together with the receipt R^* from $\mathcal{P}$. We include the description of the validation of a non-reusable service and for a reusable service (prefixed with a number of uses *m*):

Verification of a non-reusable service γ_i, service provider P_i and user $\mathcal{U}$ follow these steps:

showPASS

$\mathcal{U}$ 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 P_i: ${\mathsf{m}_1}^* = (\mathsf{m}_1, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{m}_1)))$;

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

→ **else:**

5) looks for the PASS PASS^* in its *SpentCityPASSES* database using PASS.Sn; and verifies that the ticket has not been spent:

[2]A communication channel is *resilient* if a message inserted into such a channel will eventually be delivered.

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) $Challenge^* = (Challenge, \mathsf{sk}_{\mathcal{P}i}(hash(Challenge)))$ and sends this signature to $\mathcal{U}$;
iv) asynchronously, for optimization, pre-computes $\mathsf{H_U}^{\mathsf{c}} \pmod p$;

$\mathcal{U}$ 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: $\mathsf{sk}_{\mathcal{U}}(\mathsf{pk}_{\mathcal{P}i}(\mathsf{PASS.Sn}, \mathsf{w}_3, \tau_1))$;

P_i follows the next steps:

i) computes $\alpha^{\mathsf{w}_3} \pmod p$;
ii) verifies $\alpha^{\mathsf{w}_3} \stackrel{?}{=} A_3 \cdot \mathsf{H_U}^{\mathsf{c}} \pmod p$;
→ **if verification fails:**
iii. assigns $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{01}, \tau_1)$.
iv. signs $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$ and sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathcal{U}$
→ **else:**
iii) computes $\mathsf{A}_{\mathcal{P}} = PRNG(\mathsf{h_K}) \oplus \mathsf{RI}_{\gamma_i}$, 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 $\delta_{\mathcal{T},\mathcal{P}_i}$;
iv) encrypts $\mathsf{A}_{\mathcal{P}}$ with the public key of the TTP $\mathcal{T}$: $\mathsf{pk}_{\mathcal{T}}(\mathsf{A}_{\mathcal{P}})$;
v) assigns $\mathsf{V_{succ}} = (\gamma_i, \mathsf{PASS.Sn}, \mathsf{flag}_{10}, \tau_1, \mathsf{pk}_{\mathcal{T}}(\mathsf{A}_{\mathcal{P}}))$, ($\tau_1$ is the verification timestamp). The signature is noted: $\mathsf{V_{succ}}^* = (\mathsf{V_{succ}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{succ}})))$;
vi) sends $\mathsf{m}_2 = \mathsf{V_{succ}}^*$ to $\mathcal{U}$;

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: $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$;
ii) sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathcal{U}$; indicating that the this non reusable ticket of the PASS had been already used.

showProof

$\mathcal{U}$ executes:

1) verifies P_i's signature;
→ **if $\mathsf{V_{fail}}^*$ is received or $\mathsf{V_{succ}}^*$ is not correct:**
2. $\mathcal{U}$ 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 $\mathcal{T}$;
→ **else:**
2) calculates $\mathsf{A}_{\mathcal{U}} = PRNG(\mathsf{K}) \oplus \psi_{\gamma_i,0}$, using the shared value K as seed;
3) compose the message $\mathsf{m}_3 = (\mathsf{PASS.Sn}, \mathsf{A}_{\mathcal{U}})$;
4) signs and sends it to P_i: $\mathsf{m_3}^* = (\mathsf{m}_3, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{m}_3)))$;

verifyProof

P_i follows the next steps:

1) obtains $\mathsf{PASS.Sn}$, and computes $\psi_{\gamma_i,0} = \mathsf{A}_{\mathcal{U}} \oplus PRNG(\mathsf{K})$;
2) verifies $\psi_{\gamma_i,1} \stackrel{?}{=} hash(\psi_{\gamma_i,0})$;
3) generates τ_2 and verifies it using the PASS expiry date ($\mathsf{PASS.PURdate}$, $\mathsf{PASS.EXPdate}$) and the timestamp τ_1, the value of $\mathsf{ACTdate}$ of the element ACT and $\mathsf{LIMdate}$, being $\mathsf{LIMdate} = \mathsf{ACTdate} + \mathsf{Lifetime}$;
→ **if any verification fails:**
4. assigns $\mathsf{V_{fail}} = (\gamma_i, \mathsf{PASS.Sn}, \mathsf{flag}_{03}, \tau_2, \psi_{\gamma_i,0})$.
5. signs $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$, sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathcal{U}$
→ **else:**
4) signs $\mathsf{A}_{\mathcal{P}}$ approving then the validation with timestamp τ_2: $\mathsf{R}_{\gamma_i} = (\mathsf{A}_{\mathcal{P}}, \mathsf{PASS.Sn}, \tau_2)$, and $\mathsf{R^*}_{\gamma_i} = (\mathsf{R}_{\gamma_i}, \mathsf{sk}_{\mathcal{P}}(hash(\mathsf{R}_{\gamma_i})))$;
5) stores in its *SpentCityPASSES* database: $(\mathsf{PASS}^*, \psi_{\gamma_i,0})$ and sends $\mathsf{m}_4 = \mathsf{R^*}_{\gamma_i}$ to $\mathcal{U}$;

getValidationConfirmation

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

→ **if $\mathsf{V_{fail}}^*$ is received:**
1. $\mathcal{U}$ 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 $\mathcal{T}$;
→ **else:**
1) checks the signature of $\mathsf{R^*}_{\gamma_i}$;
2) computes $\mathsf{RI}_{\gamma_i} = \mathsf{A}_{\mathcal{P}} \oplus PRNG(\mathsf{h_K})$;
3) verifies $\mathsf{h}_{\mathsf{RI}_{\gamma_i}} \stackrel{?}{=} hash(\mathsf{RI}_{\gamma_i})$;
→ **if any verification fails:**
4. $\mathcal{U}$ collects all evidence and he can rise a CLAIM by contacting with the $\mathcal{T}$, 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^*.

Verification of a *m*-times reusable service λ_i, the verification of the corresponding ticket is as follows:

showPASS

$\mathcal{U}$ 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:** $\mathcal{U}$ computes $\psi_{\lambda_i,m-1} = hash^{m-1}(\psi_{\lambda_i,0})$. Then, $\mathcal{U}$ 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[3].
3) **Case 2-Subsequent use of a m-times reusable service:** $\mathcal{U}$ 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: $\mathsf{m_1}^* = (\mathsf{m}_1, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{m}_1)))$;

verifyPASS

P_i executes:

1) verifies the PASS signature, $\mathsf{PASS.PURdate}$, and $\mathsf{PASS.EXPdate}$;
2) verifies ACT^*, and $\mathsf{PASS.ACTdate}$;
→ **if any verification fails:**
3. assigns $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{00}, \tau_1)$.
4. signs $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$ and sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathcal{U}$
→ **else:**
3) P_i looks for the PASS^* in its *SpentCityPASSES* database using $\mathsf{PASS.Sn}$; 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 $\mathcal{U}$)]:
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) $Challenge^* = (Challenge, \mathsf{sk}_{\mathcal{P}i}(hash(Challenge)))$ and sends this signature to $\mathcal{U}$;
iv) asynchronously, for optimization, pre-computes $\mathsf{H_U}^{\mathsf{c}} \pmod p$;

$\mathcal{U}$ 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: $\mathsf{sk}_{\mathcal{U}}(\mathsf{pk}_{\mathcal{P}i}(\mathsf{PASS.Sn}, \mathsf{w}_3, \tau_1))$;

P_i follows the next steps:

i) computes $\alpha^{\mathsf{w}_3} \pmod p$;
ii) verifies $\alpha^{\mathsf{w}_3} \stackrel{?}{=} A_3 \cdot \mathsf{H_U}^{\mathsf{c}} \pmod p$;
→ **if verification fails:**
iii. assigns $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{01}, \tau_1)$.
iv. signs $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$ and sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathcal{U}$
→ **else:**
iii) computes $\mathsf{A}_{\mathcal{P}} = PRNG(\mathsf{h_K}) \oplus \mathsf{RI}_{\lambda_i}$, where $PRNG(\mathsf{h_K})$ is a secure pseudorandom number generator and, $\mathsf{h_K} =$

[3]Note that $\psi_{\lambda_i,0}$ was stored by $\mathcal{U}$ in her *TicketsPASS* database at the PASS purchase phase.

$hash(\mathsf{K})$ is the seed. Note that K and RI_{λ_i} are obtained from $\delta_{\mathcal{T},\mathcal{P}_i}$;

iv) encrypts $\mathsf{A}_{\mathcal{P}}$ with the public key of the TTP $\mathcal{T}$: $\mathsf{pk}_{\mathcal{T}}(\mathsf{A}_{\mathcal{P}})$;

v) assigns $\mathsf{V_{succ}} = (\lambda_i, \mathsf{PASS.Sn}, \mathsf{flag}_{10}, \tau_1, \mathsf{pk}_{\mathcal{T}}(\mathsf{A}_{\mathcal{P}}), k_{\lambda_i}, \psi_{(\lambda,k_{\lambda_i})})$. The signature is noted: $\mathsf{V_{succ}}^* = (\mathsf{V_{succ}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{succ}})))$;

vi) sends $\mathsf{m}_2 = \mathsf{V_{succ}}^*$ to $\mathcal{U}$;

b) else[4]:

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: $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}}(hash(\mathsf{V_{fail}})))$;

ii) sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathcal{U}$;

showProof

$\mathcal{U}$ executes:

1) verifies P_i's signature;

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

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

→ **else:**

2) calculates $\mathsf{A}_{\mathcal{U}} = PRNG(\mathsf{K}) \oplus \psi_{\lambda,(k_{\lambda_i}-1)}$, using the shared value K as seed;

3) compose the message $\mathsf{m}_3 = (\mathsf{PASS.Sn}, \mathsf{A}_{\mathcal{U}})$;

4) signs and sends it to P_i: $\mathsf{m}_3{}^* = (\mathsf{m}_3, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{m}_3)))$;

verifyProof

P_i follows the next steps:

1) obtains $\mathsf{PASS.Sn}$, and computes $\psi_{\lambda,(k_{\lambda_i}-1)} = \mathsf{A}_{\mathcal{U}} \oplus PRNG(\mathsf{K})$;

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 $(\mathsf{PASS.PURdate}, \mathsf{PASS.EXPdate})$ 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 $\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$, sends $\mathsf{m}_2 = \mathsf{V_{fail}}^*$ to $\mathcal{U}$

→ **else:**

4) signs $\mathsf{A}_{\mathcal{P}}$ approving then the validation with timestamp τ_2: $\mathsf{R}_{\lambda_i} = (\mathsf{A}_{\mathcal{P}}, \mathsf{PASS.Sn}, \tau_2)$, and $\mathsf{R^*}_{\lambda_i} = (\mathsf{R}_{\lambda_i}, \mathsf{sk}_{\mathcal{P}}(hash(\mathsf{R}_{\lambda_i})))$;

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 $\mathcal{U}$;

getValidationConfirmation

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

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

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

→ **else:**

1) checks the signature of $\mathsf{R^*}_{\lambda_i}$;

2) computes $\mathsf{RI}_{\lambda_i} = \mathsf{A}_{\mathcal{P}} \oplus PRNG(\mathsf{h_K})$;

3) verifies $\mathsf{h}_{\mathsf{RI}_{\lambda_i}} \stackrel{?}{=} hash(\mathsf{RI}_{\lambda_i})$;

→ **if any verification fails:**

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

→ **else:**

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

Verification of an infinitely-reusable service ξ_i, service provider P_i and user $\mathcal{U}$ follow a verification phase similar to the verification of a non-reusable service. In this case, however, if the user presents a valid $\mathsf{R^*}_{\xi}$ and the current date is lower than $\mathsf{LIMdate}$, the service can be used again.

[4]An error has been detected, and thus the ticket is not valid. The reason of the error can be:

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

IV. CLAIMS

The *PASS Verification* protocol has been designed as a fair exchange protocol (a valid e-ticket is given in exchange for the permission to use the service). We introduce an offline TTP [15] between the user and the provider of the service in order to guarantee the fairness of the exchange. Fairness guarantees that either each party ($\mathcal{U}$ or P_i) receives the item it expects or neither party receives any additional information about the other's item. In case of exception (i.e. an unfair situation has come up and the exchange does not ends successfully for some party), any actor can arise a claim in order to restore the fairness of the exchange and preserve the security of the system.

In the *PASS Verification* there are some cases where users can receive a wrong token from the providers that can break the fairness of the system. In theses cases, users can contact the TTP to start a new claim in order to preserve the fairness of the exchange. If any claim arises, then the $\mathcal{T}$ acts as an unbiased arbiter between parties. $\mathcal{T}$ will evaluate the presented evidence and then it will side by one party depending on the result of the verifiable evidence. Thus, the involved parties have to fix the situation according to the $\mathcal{T}$ decision.

A. Non-reusable CLAIMS

Next, we are going to summarize the claims that can arise in the *PASS Verification* protocol for non-reusable services. In all cases, at the end of the verifications, $\mathcal{T}$ will publish its resolution according to them.

1) flag_{00} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{fail}}$ from P_i indicating whether any parameter of the PASS sent by $\mathcal{U}$ has not the proper form or its signature are not valid. If $\mathcal{U}$ does not agree with P_i's, then he can submit the following items to $\mathcal{T}$:

- $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{00}, \tau_1)$; $||\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$
- $\mathsf{PASS}^* = (\mathsf{PASS}, \mathsf{Sign}_{\mathcal{I}}(\mathsf{PASS}))$;

$\mathcal{T}$ can verify that the items are linked by the $\mathsf{PASS.Sn}$, next it can check the signature and the correctness of each message (including the τ_1 and the dates inside the PASS).

2) flag_{01} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{fail}}$ from P_i indicating that $\mathcal{U}$ has not been able to demonstrate the ownership of the PASS (there was a problem during the ZKP protocol run). If $\mathcal{U}$ does not agree with the P_i's message, then he can submit the following items to the $\mathcal{T}$:

- $\mathsf{pk}_{\mathcal{P}i}(\mathsf{PASS.Sn}, \mathsf{w}_3, \tau_1)$;
- $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{01}, \tau_1)$; $||\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$;
- $Challenge^*$; m_1; PASS^*;

$\mathcal{T}$ can verify that the items are linked by the $\mathsf{PASS.Sn}$, next it can check the signature and the correctness of each message. In addition, $\mathcal{T}$ asks P_i to decrypt $\mathsf{pk}_{\mathcal{P}i}(\mathsf{PASS.Sn}, \mathsf{w}_3, \tau_1)$. Then it will check whether the ZKP's response is correct or not.

3) flag_{10} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{succ}}$ from P_i indicating that the ticket is valid and has not been spent yet. However, if $\mathcal{U}$ detects that some of the parameters of this message are not correct, then he can submit the following items to the $\mathcal{T}$:

- $\mathsf{V_{succ}} = (\gamma_i, \mathsf{PASS.Sn}, \mathsf{flag}_{10}, \tau_1, \mathsf{pk}_{\mathcal{T}}(\mathsf{A}_{\mathcal{P}}))$ $||\mathsf{V_{succ}}^* = (\mathsf{V_{succ}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{succ}})))$;
- m_1; PASS^*;

$\mathcal{T}$ can verify that the items are linked by the PASS.Sn, next it can check the signature and the correctness of each message.

4) flag_{02} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{fail}}$ from P_i indicating that the ticket has been spent, thus it is not valid anymore. But, if $\mathcal{U}$ does not agree with the P_i's message, then he can submit the following items to the $\mathcal{T}$:

- $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \psi_{\gamma_i,0}, \mathsf{flag}_{02}, \tau_1, \gamma_i)$ $||\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$;
- m_1; PASS^*;

$\mathcal{T}$ can verify whether the evidence $\psi_{\gamma_i,0}$ unveiled P_i is correct or not. Thus, it is able to check if the PASS has been already spent.

5) flag_{03} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{fail}}$ from P_i indicating two possible errors: $\psi_{\gamma_i,0}$ does not match with $\psi_{\gamma_i,1}$ ($\psi_{\gamma_i,1} = hash(\psi_{\gamma_i,0})$), or τ_2 does not match the expiry date of the PASS. But, if $\mathcal{U}$ does not agree with the error message, then he can submit the following items to the $\mathcal{T}$:

- $\mathsf{V_{fail}} = (\gamma_i, \mathsf{PASS.Sn}, \mathsf{flag}_{03}, \tau_2, \psi_{\gamma_i,0})$ $||\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}}))$ $||\mathsf{m}_2 = \mathsf{V_{fail}}^*$;
- $\mathsf{m}_3 = (\mathsf{PASS.Sn}, \mathsf{A}_{\mathcal{U}})$

$\mathcal{T}$ can verify the evidence $\psi_{\gamma_i,1} \stackrel{?}{=} hash(\psi_{\gamma_i,0})$ because it has access to K (see III-A3). Also, it can verify whether τ_2 is correct or not according to the expiry date of the PASS.

6) Authorization Failure CLAIM: $\mathcal{U}$ has received a wrong R_{γ_i}. Therefore he is not able to use the service. In order to solve the situation, he can submit the following items to $\mathcal{T}$:

- $\mathsf{R}_{\gamma_i} = (\mathsf{A}_{\mathcal{P}}, \mathsf{PASS.Sn}, \tau_2)$ $||\mathsf{R^*}_{\gamma_i} = (\mathsf{R}_{\gamma_i}, \mathsf{sk}_{\mathcal{P}}(hash(\mathsf{R}_{\gamma_i})))$ $||\mathsf{m}_4 = \mathsf{R^*}_{\gamma_i}$
- $\mathsf{PASS}^* = (\mathsf{PASS}, \mathsf{Sign}_{\mathcal{I}}(\mathsf{PASS}))$;

$\mathcal{T}$ can decrypt $\mathsf{A}_{\mathcal{P}}$ evidence, thus can check whether the decrypted value RI_{γ_i} matchs $hash(\mathsf{RI}_{\gamma_i})$ stored in the PASS.

B. Reusable CLAIMS

This section summarizes the claims that can arise in the *PASS Verification* protocol for m-times reusable services.

1) flag_{00} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{fail}}$ from P_i indicating if any parameter of the PASS sent by $\mathcal{U}$ has not the proper form or its signature is not valid. If $\mathcal{U}$ does not agree with P_i's message, then he can submit the following items to $\mathcal{T}$:

- $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{00}, \tau_1)$; $||\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$
- $\mathsf{PASS}^* = (\mathsf{PASS}, \mathsf{Sign}_{\mathcal{I}}(\mathsf{PASS}))$;

$\mathcal{T}$ can verify that the items are linked by the PASS.Sn, next it can check the signature and the correctness of each message (e.g. the τ_1 is past the final expiry inside the PASS).

2) flag_{01} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{fail}}$ from P_i indicating that $\mathcal{U}$ has not been able to demonstrate the ownership of the PASS (there was a problem during the ZKP protocol run). If $\mathcal{U}$ does not agree with the P_i's message, then he can submit the following items to the $\mathcal{T}$:

- $\mathsf{pk}_{\mathcal{P}i}(\mathsf{PASS.Sn}, \mathsf{w}_3, \tau_1)$;
- $\mathsf{V_{fail}} = (\mathsf{PASS.Sn}, \mathsf{flag}_{01}, \tau_1)$; $||\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$;
- $Challenge^*$; m_1; PASS^*;

$\mathcal{T}$ can verify that the items are linked by the PASS.Sn, next it can check the signature and the correctness of each message. In addition, $\mathcal{T}$ asks P_i to decrypt $\mathsf{pk}_{\mathcal{P}i}(\mathsf{PASS.Sn}, \mathsf{w}_3, \tau_1)$. Then, it will check if the ZKP's response is correct.

3) flag_{10} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{succ}}$ from P_i indicating that the ZKP has finished successfully (i.e. $\mathcal{U}$ has proven the ownership of the PASS) and, also, P_i agrees with counter k_{λ_i}. So, the service can still be used by $\mathcal{U}$ using the $PASS$. However, if $\mathcal{U}$ detects that some of the parameters of this message are not correct, then he can submit the following items to the $\mathcal{T}$:

- $\mathsf{V_{succ}} = (\lambda_i, \mathsf{PASS.Sn}, \mathsf{flag}_{10}, \tau_1, \mathsf{pk}_{\mathcal{T}}(\mathsf{A}_{\mathcal{P}}), k_{\lambda_i}, \psi_{(\lambda,k_{\lambda_i})})$ $||\mathsf{V_{succ}}^* = (\mathsf{V_{succ}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{succ}})))$;
- m_1; PASS^*;

$\mathcal{T}$ can verify that the items are linked by the PASS.Sn, next it can check the signature and the correctness of each message.

4) flag_{04} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{fail}}$ from P_i indicating whether any parameter of the PASS sent by $\mathcal{U}$ are not correct[4]. If $\mathcal{U}$ does not agree with the P_i's message, then he can submit the following items to the $\mathcal{T}$:

- $\mathsf{V_{fail}} = (\lambda_i, \mathsf{PASS.Sn}, \mathsf{flag}_{04}, \tau_1, k_{\lambda_i}, \psi_{(\lambda,k_{\lambda_i})})$ $||\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$;
- m_1; PASS^*;

$\mathcal{T}$ can verify that the items are linked by the PASS.Sn, next it can check the signature and the correctness of each message (e.g. whether $\psi_{(\lambda,k_{\lambda_i})}$ proposed by P_i in $\mathsf{V_{fail}}$ is equal to $hash^{(k_{\lambda_i})}(\psi_{\lambda_i,0})$ or not).

5) flag_{05} *CLAIM:* $\mathcal{U}$ has received $\mathsf{V_{fail}}$ from P_i indicating two possible errors: $\psi_{\lambda,(k_{\lambda_i}-1)}$ does not match ($\psi_{\lambda,(k_{\lambda_i})} \stackrel{?}{=} hash(\psi_{\lambda_i,(k_{\lambda_i}-1)})$), or τ_2 does not match the expiry date of the PASS. But, if $\mathcal{U}$ does not agree with the error message, then he can submit the following items to the $\mathcal{T}$:

- $\mathsf{V_{fail}} = (\lambda_i, \mathsf{PASS.Sn}, \mathsf{flag}_{05}, \tau_2, \psi_{\lambda,(k_{\lambda_i}-1)}, (k_{\lambda_i}-1))$ $||\mathsf{V_{fail}}^* = (\mathsf{V_{fail}}, \mathsf{sk}_{\mathcal{P}i}(hash(\mathsf{V_{fail}})))$ $||\mathsf{m}_2 = \mathsf{V_{fail}}^*$;
- $\mathsf{m}_3 = (\mathsf{PASS.Sn}, \mathsf{A}_{\mathcal{U}})$

$\mathcal{T}$ can verify $\psi_{\lambda,(k_{\lambda_i}-1)}$ evidence, because it has access to K (see III-A3). Also, it can verifies whether τ_2 is correct or nor according to the expiry date of the PASS.

6) Authorization Failure CLAIM: $\mathcal{U}$ has received a wrong R_{λ_i} and, therefore he is not able to reach use the service. In order to solve the situation, he can submit the following items to the $\mathcal{T}$:

- $\mathsf{R}_{\lambda_i} = (\mathsf{A}_{\mathcal{P}}, \mathsf{PASS.Sn}, \tau_2)$ $||\mathsf{R^*}_{\lambda_i} = (\mathsf{R}_{\lambda_i}, \mathsf{sk}_{\mathcal{P}}(hash(\mathsf{R}_{\lambda_i})))$ $||\mathsf{m}_4 = \mathsf{R^*}_{\lambda_i}$
- $\mathsf{PASS}^* = (\mathsf{PASS}, \mathsf{Sign}_{\mathcal{I}}(\mathsf{PASS}))$;

$\mathcal{T}$ can decrypt $\mathsf{A}_{\mathcal{P}}$ evidence, thus can check whether the decrypted value RI_{λ_i} matchs $hash(\mathsf{RI}_{\lambda_i})$ stored in the PASS.

V. Security and privacy of the system

The analysis of the requeriments of the system can be performed from the security and privacy point of view and from the functionality point of view. The functional requirements, such as reduced size, flexibility, easyness of use, efficiency and so on, will be evaluated after the implementation of the system. In the other hand, we have written a formal security analysis based in the list of requirements for electronic tickets, both listed in section II-A and included in the survey [18] but due to space constrains we cannot include it in the paper.

Briefly, in terms of security, the mCITYPASS system complies with the requirements of authenticity, non-repudiation, integrity, reusability, unforgeability, fairness, non-overspending, unsplittability and exculpability. For the peculiarities of a CITYPASS system transferability is not contemplated since the PASS is intended for its usage by a single user. Moreover the reusability have to be defined for each particular service included in the PASS, with independence of the other services included in it. Unsplittability [7] property is also fulfilled by the mCITYPASS scheme, while exculpability and fairness can be assured thanks to the claims described in section IV. In terms of privacy, the user is protected due to the revocable anonymity of the system. Only the identity of fraudulent users will be disclosed.

VI. Conclusions and further works

Several proposals have been presented describing electronic ticketing systems [2], [5], [8], [14], [19], [24]. These proposals are general purpose systems or applied to transport services. In this paper we have presented an electronic ticketing system intended to be used for touristic services. Almost all big cities have inneficient CITYPASS systems usually implemented on smart cards. Our proposal is the first one that can be implemented on portable devices, such as smartphones, and is flexible enough to include reusable and non-reusable services in the same PASS. The design of the system has been performed taking into account all the security and privacy requirements described for electronic tickets, including the challenging ones (exculpability, reusability and unsplittability) resulting in a very secure and powerful system. Moreover, the dispute resolution protocol assures that all the parts are protected against other part's attacks. Finally, the system allows the user to use the system anonymously, so the privacy of the system is assured. As a future work, we want to implement the system and obtain experimental results in order to verify the usability and performance of the system. Moreover, the security and performance analysis will be presented in a formal way.

Disclaimer and Acknowledgements

This work was partially supported by the Spanish Government under AccessTur TIN2014-54945-R and Red de excelencia Consolider ARES TIN2015-70054-REDC.

References

[1] Silvia Angeloni. A tourist kit made in italy: An intelligent system for implementing new generation destination cards. pages 52: 187–209. Tourism Management.

[2] Frederik Armknecht, Alberto N. Escalante B., Hans Löhr, Mark Manulis, and Ahmad-Reza Sadeghi. *Secure Multi-Coupons for Federated Environments: Privacy-Preserving and Customer-Friendly*, pages 29–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[3] S. W. Bang, K. J. Park, W. S. Kim, G. D. Park, and D. H. Im. Design and implementation of nfc-based mobile coupon for small traders and enterprisers. In *2013 International Conference on IT Convergence and Security (ICITCS)*, pages 1–2, Dec 2013.

[4] A. Basili, W. Liguori, and F. Palumbo. Nfc smart tourist card: Combining mobile and contactless technologies towards a smart tourist experience. In *WETICE*, pages 249–254, June 2014.

[5] Sommer B. Wermuth C. M. Bhm, A. Location-based ticketing in public transport. pages 12:837–840. Proceedings of 8th International IEEE Conference on Intelligent Transportation Systems.

[6] City Cards. We love city cards, Date of access: February, 2017. http://welovecitycards.com.

[7] Liqun Chen, Alberto N. Escalante B., Hans Löhr, Mark Manulis, and Ahmad-Reza Sadeghi. *A Privacy-Protecting Multi-Coupon Scheme with Stronger Protection Against Splitting*, pages 29–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[8] Yu-Yi Chen, Chin-Ling Chen, and Jinn-Ke Jan. A mobile ticket system based on personal trusted device. *Wireless Personal Communications: An International Journal*, 40(4):569–578, 2007.

[9] J. Y. J. Chow. Policy analysis of third party electronic coupons for public transit fares. pages 66(1), 238–250. Transportation Research.

[10] Inc. City Pass, Date of access: February, 2017. http://citypass.com.

[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symposium*, 2004.

[12] S. Dominikus and M. Aigner. mcoupons: An application for near field communication (nfc). In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 2, pages 421–428, May 2007.

[13] H. C. Hsiang. A secure and efficient authentication scheme for m-coupon systems. In *2014 8th International Conference on Future Generation Communication and Networking*, pages 17–20, Dec 2014.

[14] Joanna Kos-Labedowicz. *Integrated E-ticketing System – Possibilities of Introduction in EU*, pages 376–385. Springer Berlin Heidelberg, 2014.

[15] Steve Kremer, Olivier Markowitch, and Jianying Zhou. An intensive survey of fair non-repudiation protocols. *Computer Communications*, 25:1606–1621, 2002.

[16] X. Liu and Q. l. Xu. Practical compact multi-coupon systems. In *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 3, pages 211–216, Nov 2009.

[17] Niina Mallat, Matti Rossi, Virpi Kristiina Tuunainen, and Anssi Öörni. An empirical investigation of mobile ticketing service adoption in public transportation. *Personal and Ubiquitous Computing*, 12(1):57–65, 2008.

[18] Macià Mut-Puigserver, M. Magdalena Payeras-Capellà, Josep-Llus Ferrer-Gomila, Arnau Vives-Guasch, and Jordi Castellà-Roca. A survey of electronic ticketing applied to transport. *Computers & Security*, 31(8):925 – 939, 2012.

[19] D. Quercia and S. Hailes. Motet: Mobile transactions using electronic tickets. In *1st International Conference on Security and Privacy for Emerging Areas in Communications Networks, Proceedings*, pages 374–383, Athens, Greece, Sep 2005. vol. 24.

[20] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[21] European Parliamentary Research Service. Integrated urban e-ticketing for public transport and touristic sites. 2014.

[22] William Stallings. Cryptography and network security: Principles and practice 7th ed. Pearson Education Ltd, 2017. ISBN: 13:978-0134444284.

[23] European Union. Cleaner and better transport in cities, civitas 2020, Date of access: February, 2017. http://www.civitas.eu/content/integrated-e-ticketing-system.

[24] Arnau Vives-Guasch, Magdalena Payeras-Capellà, Macià Mut Puigserver, Jordi Castellà-Roca, and Josep Lluís Ferrer-Gomila. A secure e-ticketing scheme for mobile devices with near field communication (NFC) that includes exculpability and reusability. *IEICE Transactions*, 95-D(1):78–93, 2012.