ANONYMOUS SUBSCRIPTION SCHEMES
A Flexible Construction for On-line Services Access

María Isabel González Vasco
Dep. Matemática Aplicada, Univ. Rey Juan Carlos, Madrid, Spain

Somayeh Heidarvand* and Jorge L. Villar
Dep. Matemática Aplicada IV, Univ. Politécnica de Cataluña, Barcelona, Spain

Keywords: Anonymous authentication, Blind signatures, Clone detection, Traceability.

Abstract: In traditional e-cash systems, the tradeoff between anonymity and fraud-detection is solved by hiding the identity of the user into the e-coin, and providing an additional triggering mechanism that opens this identity in case of double spending. Hence, fraud detection implies loss of anonymity. This seems to be a somewhat natural solution when universality of the e-coin is required (*i.e.*, the use of the coin is not determined at the time the coin is generated). However, much simpler protocols may suffice if we only want to prevent that payments for accessing certain services are over-used, even when users' anonymity is perfectly preserved.
In this paper we propose a simple and efficient *Subscription Scheme*, allowing a set of users to anonymously pay for and request access to different services offered by a number of service providers. In our approach, the use of the token is completely determined at issuing time, yet this final aim remains hidden to the issuing authority. Moreover, fraud detection here implies no loss of anonymity; as we make access tokens independent of the owner in a quite simple and efficient way. On the other hand, if different usages of the same token are allowed, these are fully traceable by the service providers.

1 INTRODUCTION

Anonymity in internet transactions is essential to prevent critical personal data to be inadvertently leaked to unwanted people. As an example, an eavesdropper could learn some private information about health, consumer habits or preferences of people if their identity is revealed during internet transactions. However, anonymity could be abused to make criminal acts unlinkable to individuals. To prevent such abuse, in some e-cash protocols the identity of a user can be opened under very special circumstances (*e.g.*, double spending of electronic cash).

In traditional e-coins, the tradeoff between anonymity and fraud-detection (*i.e.*, double spending or over spending) is solved by hiding the identity of the user into the coin and providing an additional triggering mechanism that opens this identity in case of double spending. Hence, fraud detection implies loss of anonymity. This seems to be a somewhat natural solution when universality of the coin is required (*i.e.*, the use of the coin is not determined at the time the coin is generated). Double spending can only be detected (yet not prevented) by the issuer (bank). Otherwise, all merchants would have to collaborate to check for the freshness of every coin.

Nevertheless, in some real life environments (*e.g.*, online games) the potential damage produced by a dishonest user is very limited, and it is often enough to guarantee some sort of "cloning detection" to prevent overuse of credit vouchers, without providing any identity-escrow mechanism. Indeed, this relaxation allows for simpler and more efficient payment schemes for many concrete applications.

1.1 Our Contribution

In this paper we describe *subscription schemes* which allow a set of users to buy access to a limited set of services, in a perfectly anonymous and efficient way. This access is paid to an issuing authority that dispenses *connection tokens*, which usage is completely

*This research is partially funded by the Spanish CRM.

González Vasco M., Heidarvand S. and L. Villar J. (2010).
ANONYMOUS SUBSCRIPTION SCHEMES - A Flexible Construction for On-line Services Access.
In *Proceedings of the International Conference on Security and Cryptography*, pages 120-131
DOI: 10.5220/0002973401200131

determined at issuing time. More precisely, tokens are differentiated in terms of their service providers and validity period (so, time is divided into different time slots). This implies that each service provider can locally and non-interactively take control on the different tokens spent in each time slot, thus rejecting any attempt of token misuse (including over use, incorrect service provider or incorrect time slot).

Following this approach, fraud-detection does not require identification of the owner, and then no loss of anonymity is implied. This will allow for a design in which tokens are independent of any private information identifying the owner in a quite simple and efficient way.

Note that it is reasonable to expect that some information about the user identity will be learned by the issuer agency (as indeed payment is a part of the token issuing protocol). However, it is our goal that this information cannot be linked either to the token itself or to the service the token is intended for. Thus, we will impose that the view of the issuing authority must be independent of the value of the issued token. As a result, no collusion of the issuer agency and one or more service providers will learn any information about the token owner.

Furthermore, payment is organized in such a way that at the end of a time slot, every service provider sends the collected tokens to the issuer to be paid for the offered service. Unused tokens can similarly be refunded to the users upon request. Thus, the subscription scheme must ensure that no collusion of users and service providers can forge new valid tokens (not issued by the agency) and they will furthermore not succeed in getting paid more than once for each issued token.

Based on well-known primitives (such as secure blind signatures and encryption schemes) we provide a new simple and practical scheme for handling access policies to on-line services. Our design basically works as follows: Users obtain from an issuing agency some tokens, consisting of a blind signature on a message including a fresh public key (for a signature scheme), the identity of a service provider and a time slot. To access the service, the user signs a random nonce, with respect to the public key contained in the token, and sends it along with the token itself to the service provider. With this simple setting we achieve:

- Perfect user anonymity with respect to the services he purchased (even when some service providers and the issuer collude).
- Unforgeability of tokens by a collusion of dishonest users and service providers.
- Undeniability of purchased services; valid access tokens cannot be repudiated by the issuing authority.
- Efficient management of tokens due to the independence of services and time slots.
- Efficient access to services for users.
- Very flexible access management for the service provider. (Token overuse is not only detected but immediately prevented by the service provider.)

Maybe the main limitation of our scheme resides in the complete traceability of the different accesses with the same token to the same service within the same time slot. However, this behavior is the desirable one when the service requires storing some settings (like preferences, history, etc.) for each (anonymized) user account.

All in all, our protocol suits many real life application scenarios, such as on-line games and on-line service subscriptions (to on-line press, digital libraries, music collections, etc.) and could also be applied to audience controls in metering schemes.

1.2 Road Map

The paper is organized as follows: we start by briefly reviewing related3 prior work in Section 2. Then, Section 3 is devoted to the introduction of what we call *Subscription Schemes*, making precise the involved entities, modeling their interaction and defining the security properties we aim at. Our basic construction is described in Section 4. In Section 5, we address some efficiency issues. We also describe some particular scenarios in which no trust on the service providers is required and some hints about how to manage different service access policies in Section 6.

Since our proposal is based on the use of a blind signature scheme, we give the necessary related definitions in Appendix 6.2.

2 RELATED WORK

Anonymity in commercial transactions (also known in some papers as untraceability) has been firstly introduced by Chaum in the seminal paper on blind signatures (Chaum, 1983). Chaum's *electronic coins* were defined as a value together with a signature from the issuing bank, which was to be withdrawn and spent by the user and subsequently deposited by the shop in the bank (thus, correctness of payment is checked on-line). In that setting, blind signature schemes are introduced as a cryptographic tool to allow the bank constructing electronic coins, in such a way that he will not be able to recognize them later.

Hence it will not be able to link a coin with the user that requested it, or identify whether two payments have been made by the same user.

Subsequent work aimed at electronic coins that could be used in an off-line setting. Namely, the shop will only deposit coins every now and then, and if a client paid with the same coin twice, his identity would be revealed. Several solutions based on RSA and Schnorr signatures can be found in (Chaum et al., 1989; Brands, 1993; Ferguson, 1994).

In some applications, total anonymity of electronic cash is not desirable (for instance, it could be used as an effective method for "whitening" black money). Several proposals for *partial* or *revokable anonymity* can be found in the literature (*e.g.*, (Camenish et al., 1997; Solms and Naccache, 1992; Jakobsson and Yung, 1996)). In these schemes anonymity may be revoked by a Trusted Third Party under certain circumstances.

Recently, some solutions in the literature with how to prove membership to a group in an anonymous way have been proposed in the context of group and ring signatures (*e.g.*, (Chang and Hwang, 2005; Fujii et al., 2007)). However, as far as we know, in that scenario no protection against double-use of access credentials has been considered. Damgård *et al.* (Damgard et al., 2006) introduced at Eurocrypt 2006 so called *unclonable group identification schemes*; which allow an honest participant to anonymously and unlinkably authenticate himself as member of a designated group. Moreover, such scheme discloses the identity of any participant that "clones" himself and connects twice with the same keying material. In their paper, Damgård *et al.* give a generic yet inefficient construction. They also describe a concrete instance, which employs some new zero-knowledge techniques. Even though the gain in efficiency is significant, still the resulting scheme is computationally rather expensive. Subsequent work of Camenish *et al.* (Camenisch et al., 2006) considers a slightly different goal; each participant should obtain, upon connection with an issuer/authority, enough information to connect *k* times to a service (anonymously and unlinkably). Again, overusing this private connection information leads to the identification of the fraudulent participant. Their solution, though more practical than that of Damgård *et al.*, is still rather costly—in particular if we look at the number of operations a user has to preform each time he connects—.

Closer to our work, recently, Blanton (Blanton, 2008) proposes a subscription scheme which is similar in spirit to our construction; however, no separation between service provider and issuer is made, which in particular forces the service provider to store all access tokens ever presented. Moreover, it is computationally more costly, as each access involves an interactive zero knoledge proof (this however could, as noted by the author, maybe be replaced using recent work of Groth and Sahai (Groth and Sahai, 2008)). Similarly, Razman and Ruhl (Ramzan and Ruhl, 2000) put forward a model for subscription-based services which is however less flexible than ours; at it, each user obtains a fixed number of accesses to the service, but without expiration date.

3 SUBSCRIPTION SCHEMES

We start by giving a formal description of what we call a *Subscription Scheme*.

3.1 Involved Entities

Our *subscription scheme* involves different entities, modelled by probabilistic polynomial-time interactive Turing machines:

- a finite set of *service providers*, $\mathcal{SP} = \{\mathsf{SP}_1, \ldots, \mathsf{SP}_n\}$, each of them offering a concrete service managed according to their own policy. This policy must specify the duration of subscriptions to this service, using as time reference different time slots and possibly, also session identifiers distinguishing different sessions per slot. We assume this providers will never deny access upon request with a valid token.[2]
- a finite set of *users*, $\mathcal{U} = \{\mathsf{U}_1, \ldots, \mathsf{U}_m\}$, which may subscribe to any of the services above,
- an *issuing authority* IA which role is to publish and certify all information about the service providers, and dispense subscription tokens to users upon request (and payment).
- a *trusted third party* TP which will be invoked by a user in case he wants to be refunded for an unused token. This trusted party can also be used to guarantee the fairness of all paying protocols in the system. We may assume the TP is connected with each user via a private and authentic channel.

3.2 Scheme Syntax

Now, the interaction between these entities is specified by the following algorithms and protocols, which define the *subscription scheme*. Here, for simplicity

[2]This is quite a natural semi-honesty assumption, as it is in their own interest to gain customer loyalty. See Section 6 for some ways to remove this assumption.

we assume that every token allows the user for a single access to a service. For other access policies (*e.g.*, multiple accesses with the same token) see Section 6.

Start-up Algorithms. They are only run during a set up phase, and provide all involved entities, on the input of the security parameter and possibly some other system parameters, with all the public/private key pairs needed for the scheme.

- `IAKeyGen.` Run once by the IA; it outputs all public key/secret key pairs needed for the protocol
- `SPKeyGen.` Run once by each service provider SP_j; it outputs all public key/secret key pairs needed for the protocol,
- `PublishCatalogue.` Run once by the IA; on the input of the public keys and service information of the service providers it outputs an authenticated catalogue (*e.g.*, signed by the IA), including at least all service providers' identifiers and public keys, as well as the service descriptions and conditions of use.

Subscription Protocols. We assume that the catalogue of services and the current time slot are always included as common inputs to all protocols. We also assume that all entities are supposed to be able to verify the authenticity of all public keys. Actually, only the public key of the IA needs to be certified externally. [3]

- `VerifyToken.` Run by any party, on the input of a token x a service provider identifier SP and a time slot t it outputs a single bit indicating the validity of x. This auxiliary algorithm will be used in the protocols described below.
- `ObtainToken.` This protocol is run by a user U and the issuing authority IA. User's private input will include a service provider's name SP and a time slot identifier t' (not necessarily the current one). As private output, U will either receive an error message $\perp$ or a valid token to access the service offered by SP on time slot t', according to the service provider's particular access policy. To ensure this, U might execute the `VerifyToken` at some point during the protocol execution.

  Typically, an optimistic fair e-cash protocol is involved in this step since at this point the user pays for the service requested. This protocol requires the intervention of a Trusted Party, in order to guarantee its fairness. At this, some information about the identity of the user might be leaked, but the IA shall get no information at all about SP or t'.

  Note that the IA will always get the information corresponding to the amount paid by the user in each transaction, but we want that this is the only information he may have in order to link user identities with requested services. Bearing this in mind, in the sequel we may assume all services offered at a given time slot have the same price.
- `AccessService.` This protocol is run by a user U and a service provider SP. User's private input includes the token, and SP's private input is sk_{SP}. User U requests access to the service offered by SP. He gets as output a denial or acceptance message, depending on the validity of the token, and is or not allowed into the service accordingly. As we already noted, tokens recognized as valid will be always accepted by SP. At this, the private output to SP will include some information about U's token, which, if required, could be used as a proof of service in front of the Trusted Party.

Payment Protocols.

- `Pay.` This protocol is invoked by each SP at the end of every time slot, and involves him and the IA. SP sends part of the private outputs collected after successful `AccessService` executions, including a list of the collected tokens, to the IA, to be paid for the offered service. At the end of the protocol, SP gets paid for the list of tokens and the IA keeps his private output as a receipt of payment, typically involving some function of SP's private keying material and the tokens. Eventually, IA could deny payment. Namely, whenever SP tries to execute the protocol twice in the same time slot, or if some of the tokens are invalid or have been refunded. An optimistic fair e-cash protocol is used here, and the same Trusted Party as above is used to guarantee the fairness.
- `Refund.` A user U executes this protocol with the Trusted Party and possibly SP and IA. U's private input includes an unused token, valid for the current time slot and service provider SP. If the Trusted Party finds that the token is valid and unused, then the user gets refunded (from IA but via the Trusted Party) for his payment. Both SP and IA will get payment receipts as private output, which SP will use to reject any further attempt to use the refunded token and IA will use to prove the Third Party that the token has been already refunded. Notice that we prefer not to rely on the state of the Third Party. Unused tokens not claimed for refund by the user are on the benefit of the IA.

[3] Note that service providers' public keys are included in the catalogue of services, thus they are automatically certified by the IA.

3.3 Security Model

We aim at providing the following properties:

Correctness. If all the involved entities act honestly then:

- Every service provider SP will grant access to any user U in the execution of `AccessService` within a time slot t, whenever U uses as private input a token obtained from the execution of `ObtainToken` for service provider SP and t.
- In all executions of `Pay`, IA will accept and refund all tokens collected by SP for a given time slot.

Fairness for the User U^*. Recall that, by assumption, a service provider SP will deny service to U^* only on input of an invalid token. An adversary corrupting all service providers, any set of users (not including the target user U^*) and the IA, has only negligible probability of winning the following game: U^*, who acts honestly, runs a polynomial number of instances of the protocol `ObtainToken` to get tokens for some service providers and time slots. Concurrently, U^* runs a polynomial number of instances of `AccessService` with some of the service providers, and also runs `Refund` with the Trusted Party giving as private input valid tokens rejected by service providers (this can only happen in case the adversary was able to construct the same token and used it before, exhausting its validity). The adversary wins the game if for a valid token x, a service provider denies access to U^* on input x, and moreover, the Trusted Party rejects U^*'s execution of `Refund` against that service provider on the same token x.

Fairness for the Service Provider SP^*. Basically, we demand that a service provider will always be paid for all services offered within a given time slot. This is formalized in the following game:

An adversary corrupting a set of users, some service providers (others than SP^*) and the IA, has negligible probability of winning the following game: Some corrupt and uncorrupt users run several instances of `ObtainToken`, `AccessService` with SP^*, and of `Refund` against SP^*. Moreover, SP^* runs several instances of `Pay` (each one at the end of a different time slot). The adversary wins the game if, impersonating the IA, he denies payment to SP^* in a `Pay` execution, and also convinces the Trusted Party that he already paid SP^* in that time slot, or that some of SP^*'s tokens are invalid or have been refunded.

Fairness for the Issuing Authority IA. Consider an adversary corrupting a set of users and some (possibly all) service providers. Let n_t be the number of tokens sold by the IA until the end of time slot t, and let n'_t the total number of tokens paid (directly by an execution of `Pay` or forced by the Trusted Party in `Refund`) by the IA in all time slots t' such that $t' \leq t$. Then, assuming that a polynomial number of concurrent executions of `ObtainToken`, `AccessService`, `Pay` and `Refund` on adaptively chosen inputs occur, the probability that $n'_t > n_t$ is negligible.

Essentially, fairness for the IA means that the only valid tokens in the system are the ones generated in a successful execution of `ObtainToken`, and that the IA will never pay twice for a given token. The first condition can be seen as a kind of token unforgeability, while the second requirement relies on the fairness of `Refund` and `Pay` protocols, and on the fact that tokens are bound to specific service providers and time slots.

Anonymity for User's Services. Consider the following indistinguishability game between an adversary $\mathcal{A}$, corrupting all parties (*i.e.*, the IA, all users and all service providers) in the system, and a challenger $\mathcal{C}$.

- $\mathcal{A}$ runs `Setup` and sends to $\mathcal{C}$ all the public information about the users, the service providers and the IA. During the whole game $\mathcal{A}$ may execute polynomially many instances of the `ObtainToken` and `AccessService` protocols. Notice that, in particular $\mathcal{A}$ learns the user's private output of `AccessService`.
- $\mathcal{A}$ chooses two (possibly equal) service providers' identities, SP_0 and SP_1, and two (possibly equal) user's identities, U_0 and U_1, and sends the choice to $\mathcal{C}$ along with the internal state (including all the secret information) of U_0 and U_1.
- $\mathcal{C}$ flips a fair coin $b \in \{0,1\}$ and prepares himself to run two (possibly concurrent) instances of `ObtainToken`, one as U_0 and the other as U_1, where $\mathcal{A}$ acts as the IA. To that end, $\mathcal{C}$ marks the protocol instance corresponding to U_0 as the target one, and uses as private input (SP_b, t), where t is the only time slot is considered in this game. The other instance's private input is (SP_{1-b}, t). If $\mathcal{C}$ obtains as outputs two valid tokens, we denote by x_b the one from the target instance of `ObtainToken`, and the other one by x_{1-b}.
- Once the two instances of `ObtainToken` terminate, if they were both successful $\mathcal{C}$ (concurrently) runs two instances of `AccessService`, one for token x_0 with $\mathcal{A}$ acting as SP_0, and the other for token x_1 and service provider SP_1.

  Otherwise, if $\mathcal{C}$ failed to obtain the two valid tokens (even if he got one), he does not run any instance of `AccessService`.

- Eventually, $\mathcal{A}$ ends the game outputting a bit b'.

The probability that $b' = b$ (case in which $\mathcal{A}$ wins the above game) should be non-negligibly greater that $1/2$.

Although the above is only one of the many possible indistinguishability-like definitions related to the anonymity of service, it can be shown that this notion implies the most general possible definition of anonymity. Namely, from the information available to the IA from `ObtainToken` instances, and to the service providers from `AccessService` instances, no polynomial time adversary can distinguish any two possible matchings between both sets of instances.

4 A BASIC SCHEME

The basic scheme uses a public-key encryption scheme *ENC*, a blind signature scheme *BSig* (for a summary of the definition and security of blind signatures, see Appendix 6.2), and basic (general purpose) signature scheme *Sig*. The *BSig* protocol is linked to a optimistic fair e-cash protocol in order to guarantee that a user gets a valid blind signature if and only if he pays to the signature issuer. This can be typically done by using the e-cash protocol to fairly send the last signer's message in the blind signing protocol. We assume that in case the user does not pay the signer then he does not receive the last message, so no blind signature is generated. Conversely, the user will not pay if the verification of the blind signature fails. To name this dedicated combination of *BSig* and a fair e-cash protocol, we will often refer to the *modified blind signature scheme*. Our Basic Construction is explained below:

Set Up. Keys for the IA and all service providers are generated and distributed:

- Each service provider SP_j holds a key pair $(pk_{\mathsf{SP}_j}, sk_{\mathsf{SP}_j})$ for the encryption scheme *ENC*, and another key pair for the signature scheme *Sig*.
- IA generates signing keys $(pk_{\mathsf{IA}}, sk_{\mathsf{IA}})$ for *BSig*. It also signs and publishes the `catalogue`.
- Each SP maintains a list L_{SP} of accepted tokens [4]. Also, IA and each SP maintain a list of tokens paid for through `Refund` for the current time slot (denoted, respectively R_{IA} and R_{SP}).

[4]Recall that in the above we are assuming for simplicity the access policy to be "one access per token", otherwise this lists would be configured fitting each concrete access policy.

Obtain Token. User U wants to buy access to SP's service in (a future) time slot t.

1. U generates a fresh key pair (y, s) for the basic signature scheme *Sig*.
2. U obtains from IA a valid[5] blind signature $\sigma = \mathtt{BlindSig}(y||\mathsf{SP}||t)$ and pays for it, by means of the modified blind sign algorithm.
3. U stores the token $x = (y, \mathsf{SP}, t, \sigma)$ and s until the end of slot t.

Verify Token. Given a token $x = (y, \mathsf{SP}, t, \sigma)$, any party can verify its correctness by just verifying that σ is a valid blind signature of $m = y||\mathsf{SP}||t$.

Access Service. User U requests access to the service SP on time slot t :

1. U sends an access request message to SP, involving a random nonce ρ.
2. SP generates a random nonce α and forwards it to U.
3. U computes $c = ENC_{\mathsf{SP}}(y||\sigma||\tilde{\sigma})$, where $\tilde{\sigma} = Sig_s(\alpha||\rho)$, and sends c to SP.
4. SP decrypts c and parses y, σ and $\tilde{\sigma}$.
5. SP checks that σ is a valid signature of $y||SP||t$ and that $\tilde{\sigma}$ is a valid signature of $\alpha||\rho$ with verification key y.
6. SP also checks that σ is not in the refunded token list R_{SP}.
7. SP looks at the access table for previous usages of y [6] and applies the service terms of use to decide acceptance.
8. If all checks are OK, SP allows U into the server and adds a new row $(\alpha||\rho, y, \sigma, \tilde{\sigma})$ to the access table L_{SP}.

Pay. At the end of the time slot, each SP runs the following protocol:

1. SP sends the list of collected (*i.e.*, valid and not refunded) (y, σ) to IA.
2. IA checks whether he paid SP before in the current time slot. If not, IA checks the validity of all the items in the list for the current time slot, and that none of them have been refunded (looking them at R_{IA}), and pays SP for them via the fair e-cash protocol.

[5]Here, we impose U does have the ability to actually check the validity of the received token, as it is explicited later in `VerifyToken`.

[6]Checking for σ would be not enough unless the blind signature is strongly unforgeable, as we need that the adversary cannot produce produce a new signature pair (m, σ), even having different signatures on m at hand.

3. IA gets as a receipt SP's signature on the time slot identifier t, and keeps it until the beginning of next time slot.
4. SP resets his access table L_{SP} and the refund table R_{SP}, and enters in a `lock` state until the beginning of the next time slot.

Refund. User U asks the Trusted Party for an unused token refund.

1. U sends TP the (presumably) unused token $x = (y, SP, t, \sigma)$.
2. TP checks the validity of σ and asks SP for a proof of usage or previous refund.
3. If not locked, SP checks for usages of y in table L_{SP} and sends the corresponding entry $(\alpha||\rho, \tilde{\sigma})$, if it exists. He also checks if σ is in table R_{SP} and if so, sends the corresponding refund receipt.
4. If in either case TP accepts SP's proof (or if SP in locked), then TP aborts the protocol.
5. Otherwise, TP asks IA for refund on (y, SP, t, σ).
6. If after looking at R_{IA}, IA sends a receipt of previous refund on that token, then TP aborts.
7. Otherwise, TP sends a receipt (TP's signature on $\texttt{'refunded'}||t||SP||\sigma$) to both SP and IA, and sends back the cash to U.
8. SP and IA add σ and the refund receipt to the corresponding refund lists R_{SP} and R_{IA}.

4.1 Formal Analysis

Let us now argue our generic construction fulfils the properties listed in Subsection 3.3. At this, we are assuming that the underlying blind signatures scheme *BSig* has the blindness and non-forgeability property, as defined in Appendix 6.2. Moreover, we assume the encryption scheme *ENC* to be IND-CCA secure. The basic signature scheme *Sig* is assumed to be existentially unforgeable under chosen message attacks. Finally, we assume the fairness of the optimistic e-cash protocols used in `ObtainToken` and `Pay`.

Correctness. It follows trivially from the correctness of the involved tools *BSig*, *Sig* and *ENC*, and the e-cash protocols.

Fairness for the User U^*. Note that the adversary will not be able to replay an eavesdropped connection message c from a previous connection, as c involves a signature of the nonce α that can only be used once. Therefore, the adversary wont succeed in a strategy of "exhausting" the usage of a token legitimately obtained by U^*.

As a result, the only case in which fairness for user U^* may be violated is that in which for a valid token x, a corrupt service provider denies access to U^* on input a legitimate c constructed from x and, moreover, the Trusted Party rejects U^*'s execution of `Refund` against that service provider on that same token x.

However, the Trusted Party rejects U^*'s execution of `Refund` only if the adversary $\mathcal{A}$ defined in Section 3.3 shows him a valid pair $(\alpha||\rho, \tilde{\sigma})$, where α is a session identifier and $\tilde{\sigma}$ is a basic signature on $\alpha||\rho$, with respect to the verification key y. But this is only possible if either U^* computed $\tilde{\sigma}$ (so he indeed accessed the service) or $\mathcal{A}$ forged that signature.

Fairness for the Service Provider SP^*. Suppose that an honest service provider SP^* and an adversary $\mathcal{A}$ are playing the game corresponding to the present security notion, as described in Section 3.3. Let $L_{SP^*} = \{(y_k, \sigma_k, \alpha_k||\rho_k, \tilde{\sigma}_k)\}$ be the contents of SP^*'s access table at the end of a specific time slot t. Notice that each σ_k is a valid blind signature on $m_k = y_k||SP^*||t$, and all m_k are different. At the end of the time slot, SP^* runs `Pay` with the adversary, who acts as the IA, for list L_{SP^*}.

Assume that $\mathcal{A}$ cheats SP^* and denies payment. Now SP^* complains to the Trusted Party, by sending him the list L_{SP^*}. As SP^* acts honestly, the Trusted Party is convinced about the validity of the collected tokens. Next, the Trusted Party asks $\mathcal{A}$, who acts as the IA, for both a list of receipts for tokens in L_{SP^*} which have been refunded, and a payment receipt for SP^* and current time slot. Since SP^* acts honestly, there are no unused tokens in L_{SP^*}. Hence, the only way $\mathcal{A}$ can show a refund receipt for a token in L_{SP^*} is by forging a signature on the token on behalf of the Trusted Party. Indeed, no used token can be refunded, since during the execution of `Refund`, the Trusted Authority asks SP^* for a proof of usage of the token, and SP^* answers with a valid pair $(\alpha||\rho, \tilde{\sigma})$, so the Trusted Party denies refunding.

On the other hand, $\mathcal{A}$ cannot show a payment receipt for the current time slot, and thus the Trusted Party forces him to pay SP^* for all tokens in L_{SP^*}. Indeed, due to the fairness of the e-cash protocol in `Pay`, $\mathcal{A}$ can only show a payment receipt if he forged one (*i.e.*, he forged a signature by either SP^* or the Trusted Party) or if he successfully ran `Pay` with SP^* before. But the last situation is impossible, as an honest SP^* runs `Pay` at most once per time slot.

Fairness for the Issuing Authority IA. Consider a successful adversary $\mathcal{A}$ who plays the game defined in Section 3.3. Then, we show a forger $\mathcal{F}$, who internally uses $\mathcal{A}$, winning the blind signature unforgeability game against a challenger $\mathcal{C}$, with a non-negligible

probability.

Firstly, the challenger $\mathcal{C}$ generates, according to the specification of the blind signature scheme, the system parameters and the public key pk_{BSig}, and sends them to a forger $\mathcal{F}$. Next, $\mathcal{F}$ completes the public parameters of the subscription scheme (including the public key of the Trusted Party) and the public key of the (honest) IA, and sends this information to $\mathcal{A}$. Now $\mathcal{A}$ computes and sends to $\mathcal{F}$ the set of public keys of the service providers, and also a description of the corresponding services. $\mathcal{F}$ compiles and signs the catalogue of services and send it back to $\mathcal{A}$.

Now $\mathcal{A}$, acting as a (dishonest) user, concurrently runs polynomially many instances of BlindSig with $\mathcal{F}$ acting as the IA. $\mathcal{A}$ can also run a polynomial number of instances of the protocols Refund and Pay. Here, $\mathcal{A}$ takes the roles of both the users and the service providers, while $\mathcal{F}$ acts as both the IA and the Trusted Party.

During the game, $\mathcal{F}$ maintains a list of all valid pairs $(m_k = y_k||\mathsf{SP}_k||t_k,\sigma_k)$ of blind signatures and messages collected in all executions of Refund and Pay. As a honest IA he also maintains lists of refunded and paid tokens, and the corresponding receipts, for each service provider, which are needed in a proper execution of those protocols.

Eventually, $\mathcal{A}$ ends the game (with a non-negligible probability of having been paid for more tokens than there were bought). Finally, $\mathcal{F}$ sends $\mathcal{C}$ the list of collected message/signature pairs, and ends the game. Here we assume that $\mathcal{F}$ maintains the list in such a way that all messages in it are different, and that all signatures are valid.

Now, let us see that $\mathcal{F}$ will only pay $\mathcal{A}$ for valid tokens, and he will never pay twice for the same token. Indeed, in both protocols Refund and Pay the IA checks the validity of the token (*i.e.*, the validity of the blind signature) before paying. On the one hand, $\mathcal{F}$ maintains a list of refunded tokens, so that any repeated execution of Refund is rejected; and this list is also used to check for duplicates in Pay. Since $\mathcal{F}$ only accepts a single execution of Pay per service provider and time slot, no token can be paid more than once[7].

Finally, due to the fairness of ObtainToken, the only executions of BlindSig accepted by $\mathcal{F}$ come from executions of ObtainToken accepted by $\mathcal{F}$ (*i.e.*, paid by $\mathcal{A}$). Hence, whenever $\mathcal{A}$ is successful, the number of executions of BlindSig accepted by $\mathcal{F}$ is less than the number of message/signature pairs outputted by $\mathcal{F}$, thus breaking the unforgeability of the blind signature scheme.

[7]Reusing a blind signature for two service providers would mean breaking the unforgeability of the signature scheme.

Anonymity for User's Services. Given a successful adversary $\mathcal{A}$ against the anonymity of the subscription scheme, we show another adversary $\mathcal{B}$ who breaks the blindness of the blind signature scheme by internally using $\mathcal{A}$. Let $\mathcal{C}$ be the challenger for $\mathcal{B}$ in the blindness game.

Firstly, $\mathcal{C}$ generates the system parameters of the blind signature scheme and gives them to $\mathcal{B}$. $\mathcal{B}$ completes the public parameters with the system parameters of the other components in the anonymous subscription system, and send them to $\mathcal{A}$. Then $\mathcal{A}$ generates the public output of the Setup protocol (*i.e.*, public keys for all entities including the public key for the blind signature pk_{BSig} and the signed catalogue of services) and sends it to $\mathcal{B}$. Now, $\mathcal{A}$ selects the target identities: SP_0, SP_1 and U_0, U_1 and sends them to $\mathcal{B}$ along with the internal state of U_1 and U_2. Notice that the internal states in particular include the secret information about user's identities, needed in the e-cash protocol. After verifying the information received from $\mathcal{A}$, $\mathcal{B}$ forwards pk_{BSig} to $\mathcal{C}$. $\mathcal{B}$ also generates two key pairs for the basic signature scheme (s_0,y_0) and (s_1,y_1), and sends $m_0 = y_0||\mathsf{SP}_0||t$ and $m_1 = y_1||\mathsf{SP}_1||t$ to $\mathcal{C}$, where t is the descriptor of the current time slot.

Now $\mathcal{C}$ flips a fair coin b and starts two instances of BlindSig on m_b and m_{1-b}, notifying $\mathcal{B}$ that the former is the target one. For each instance, $\mathcal{B}$ executes ObtainToken with $\mathcal{A}$ as the IA in the following way: $\mathcal{B}$ forwards all messages corresponding to the signing protocol from $\mathcal{C}$ to $\mathcal{A}$ and from $\mathcal{A}$ to $\mathcal{C}$, and uses the corresponding identity (U_0 for the target instance, and U_1 for the other one) in the e-cash part of the protocol. $\mathcal{B}$ also informs $\mathcal{A}$ that the instance using U_0's identity is the target one.

If at the end of the protocols $\mathcal{C}$ gets two valid blind signatures: σ_0 on $m_0 = y_0||\mathsf{SP}_0||t$ and σ_1 on $m_1 = y_1||\mathsf{SP}_1||t$, then he sends (σ_0,σ_1) to $\mathcal{B}$. Otherwise, $\mathcal{C}$ sends $\perp$ to $\mathcal{B}$.

In the first case, as $\mathcal{B}$ holds valid tokens $x_0 = (y_0,\mathsf{SP}_0,t,\sigma_0)$ and $x_1 = (y_1,\mathsf{SP}_1,t,\sigma_1)$, he runs two instances of AccessService: one for x_0 with $\mathcal{A}$ acting as SP_0, and the other for x_1 with $\mathcal{A}$ acting as SP_1. This means that $\mathcal{A}$ receives encryptions of both $(y_0||\sigma_0||\tilde{\sigma}_0)$ and $(y_1||\sigma_1||\tilde{\sigma}_1)$, for valid $\alpha_0||\rho_0$ and $\alpha_1||\rho_1$, along with valid basic signatures of them, $\tilde{\sigma}_0$ and $\tilde{\sigma}_1$, for verification keys y_0 and y_1, respectively. In the second case, no instance of AccessService is executed. In both cases, $\mathcal{A}$ eventually ends the game by outputting a guess bit b', which is forwarded to $\mathcal{C}$ by $\mathcal{B}$.

It is straightforward to see that $\mathcal{B}$ perfectly simulates a challenger for $\mathcal{A}$ in the anonymity game. So $\mathcal{A}$ wins the game with a non-negligible probability,

Table 1: Efficiency comparison between Camenisch *et al.* and our scheme, measured in number of exponentiations.

	ObtainToken		AccessService	
	User	Issuer	User	Service Prov.
Camenish *et al.* (Camenisch et al., 2006)	3	3	13	7
Ours	3	1	2	5

which is equal to the probability that $\mathcal{B}$ wins the blindness game.

5 EFFICIENT INSTANCES

In the previous sections a generic flexible anonymous subscription scheme has been presented. Here we go further in the efficiency analysis, roughly sketching the cost of concrete instantiations. To implement the scheme we propose using RSA blind signature that is fast and efficient for ObtainToken and the hashed ElGamal signature (as modified by Pointcheval and Stern (Pointcheval and Stern, 1996)) as the basic general purpose signature scheme *Sig* used in AccessService. ElGamal signing requires 1 exponentiation and verification requires 3. Furthermore, ElGamal key generation (which is required every time a token is generated) only requires one exponentiation. As IND-CCA encryption scheme *ENC*, we choose RSA OAEP+ (Shoup, 2008). The cost of encryption and decryption is just one exponentiation.

The first RSA blind signature was introduced in (Chaum, 1981) but is not secure. The Hashed RSA blind signature, which is secure in the random oracle model, is used instead. It works as follows: Assuming the usual RSA key generation, to get a blind signature on the message m, a receiver chooses a random value r relatively prime to N, computes $M = H(m)r^e$, where H is a suitable hash function, and sends it back to the signer. Then the signer computes $\sigma' = M^d = H(m)^d r$. The blind signature is computed by the receiver as $\sigma = \sigma' r^{-1}$, and it can be verified by the equation $\sigma^e = H(m)$.

Now we compare our protocol with the one by Camenisch *et al.* (Camenisch et al., 2006)[8] looking at the efficiency of the corresponding algorithms for buying tokens and connecting to the services. The comparison is summarized in Table 1.

Their ObtainToken protocol requires 6 exponentiations (3 performed by the user and 3 by the issuer). Using RSA blind signature, the complexity of obtaining a token in our proposal is basically computing 4 exponentiations (3 by the user and 1 by the issuer), which is more efficient. However Hashed RSA blind signature is known to be secure only in the random oracle model, though no known attack against it in the standard model is known.

Compared to ours, the protocol Show of Camenisch *et al.* — which is the most efficient, up to our knowledge, proposed so far — calls for 13 exponentiations from the user and 7 from the service provider, when the user connects to a service, while in our AccessService protocol only 2 exponentiation is computed by the user, and 5 exponentiations are performed by the service provider, what is far more efficient. This makes our protocol completely suitable in most practical scenarios.

6 EXTENDED FEATURES

6.1 Multiple Accesses per Token

Our description of AccessService can be easily modified to provide full flexibility of the service providers policy. Multiple accesses per token can be implemented if the Service Provider allows more than one record per token in the access table. At this, further precautions should be taken in order to prevent replay attacks, *e.g.*, we can add some structure to the nonce α. Namely, α may be the concatenation of a constant part α_0 and an access counter α_1. Then SP will only accept an access attempt for a signed nonce $\alpha_0||\alpha_1$, with $\alpha_1 > 0$, if a previous usage of the token shows the value $\alpha_0||\alpha_1 - 1$. It is straightforward for the SP to apply a limit in the number of accesses per token based on the stored value of α_1. Actually, SP can save memory if he stores only the last usage of each token.

Also, timing information can easily be added to the access table in order to apply more complex access policies involving both the number of accesses and the total access time, or the time elapsed from the first access.

On the other hand, if the service is configured in different sessions (*e.g.*, sub-services or groups) per time slot among which users may freely chose, then a (public) session identifier sid can be appended to the nonce α.

Obviously, in case of multiple accesses per token, the protocol Refund should be refined depending on the

[8]This scheme is significantly more efficient than that of Damgård *et al.* (Damgard et al., 2006).

concrete policy. For instance, the Trusted Party can consider a token unused if no access to the service have been given for that token or one may impose that tokens may be refunded as long as they are not exhausted. Additionally, partial refunds (*i.e.*, refund of the estimated unused part of a token) could be considered. However, this variant has a high cost in terms of efficiency, as the `Refund` protocol (which is likely to be very costly) will presumably be executed many times.

6.2 Removing Trust on SP

In the basic definition of `AccessService` we assumed that a Service Provider never denies access to the service if the user shows a valid unused token. However, dropping this assumption may make sense in settings in which client loyalty is not valuable; like services that are only required once and for which potential clients are not in touch with former users. At this, a dishonest SP could collect a valid token and deny access to the user. Then nobody can prevent SP to include this actually unused token in the `Pay` protocol. Actually, the Trusted Party should not accept any complaint from a user, since a dishonest user could complain just to be refunded on a used token.

In some settings this problem can be circumvented with a small overhead: if, for instance, the service consists of a user connected to a resource (*e.g.*, game, multimedia streaming, chat room, ...) for a long period of time. In such scenario the user can be requested to send his token and a signature on an incremental nonce, as explained above, at a fixed and reasonable rate (say, once every minute). In the worst case, if the Service Provider interrupts the service then he can only prove to the Trusted Party that the used got access during one more minute than the actual access time, which is not a great deal in most applications. Moreover, a user cannot ask for refund on more than the unused time, since the SP holds a user's signature on the nonce used in the last access.

REFERENCES

Blanton, M. (2008). Online subscriptions with anonymous access. In *Proceedings of the 2008 ACM Symposium on Information, computer and communicatinos security*, pages 217–227.

Brands, S. A. (1993). Untraceable Off-Line Cash in Wallets with Observers. In *CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318. Springer.

Camenisch, J., Hohenberger, S., Kohlweiss, M., Lysyanskaya, A., and Meyerovich, M. (2006). How to win the clone wars:
efficient periodic n-times anonymous authentication. Cryptology ePrint Archive, Report 2006/454. http://eprint.iacr.org/.

Camenish, J., Maurer, U., and Stadler, M. (1997). Digital Payment Systems with Passive Anonymity-Revoking Trustees. *Journal of Computer Security*, 5(1):254–265.

Chang, C.-C. and Hwang, T. (2005). Anonymous proof of membership with ring signature. In *Proceedings of the 2005 IEEE International Conference on Electro Information Technology*, pages 5–9.

Chaum, D. (1981). Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88.

Chaum, D. (1983). Blind Signatures for Untraceable Payments. In *CRYPTO 88*, pages 199–203.

Chaum, D., Fiat, A., and Naor, M. (1989). Untraceable Ellectronic Cash. In *CRYPTO 1988*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer.

Damgard, I., Dupont, K., and Pedersen, M. (2006). Unclonable group identification. In *Proceedings of EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 555–572. Springer.

Ferguson, N. (1994). Single term off-line coins. In *Proceedings of EUROCRYPT 1993*, volume 765 of *Lecture Notes in Computer Science*, pages 318–328. Springer.

Fujii, A., Ohtake, G., Hanaoka, G., and Ogawa, K. (2007). Anonymous authentication scheme for subscription services. In *Proceedings of KES 2007/WIRN 2007*, volume 4694 of *Lecture Notes in Artificial Intelligence*, pages 975–983. Springer.

Groth, J. and Sahai, A. (2008). Efficient non-interactive proof systems for bilinear groups. In *Proceedings of EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer.

Jakobsson, M. and Yung, M. (1996). Revokable and versatile electronic money. In *Proceedings of the 3rd CCCS*, volume 765, pages 76–87. ACM Press, New York.

Juels, A., Luby, M., and Ostrovsky, R. (1997). Security of blind digital signatures. In *Proceedings of CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164. Springer.

Okamoto, T. (2006a). Efficient blind and partially blind signatures without random oracles. In *Procceedings of the Third Theory of Cryptography Conference, TCC 2006*, volume 3876, pages 80–99. Springer-Verlag.

Okamoto, T. (2006b). Efficient blind and partially blind signatures without random oracles. Cryptology ePrint Archive, Report 2006/102. http://eprint.iacr.org/.

Pointcheval, D. and Stern, J. (1996). Provably secure blind signature schemes. In *Proceedings of ASIACRYPT 1996*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer.

Pointcheval, D. and Stern, J. (2000). Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396.

Ramzan, Z. and Ruhl, M. (2000). Protocols for anonymous subscripton services. Unpublished manuscript. At the time of writing, available at http://people.csail.mit.edu/ruhl/papers/drafts/subscription.pdf/.

Shoup, V. (2008). OAEP reconsidered. *Journal of Cryptology*, 15(4):223–249.

Solms, S. and Naccache, D. (1992). On Blind Signatures and Perfect Crimes. *Computers & Security*, 11:581–583.

APPENDIX

Blind Signature Schemes

The security of Blind Signatures Schemes was formalized in (Pointcheval and Stern, 1996; Juels et al., 1997). Here we follow the notation and terminology of (Juels et al., 1997), however; the definition of blindness below is taken from (Okamoto, 2006a; Okamoto, 2006b)[9].

Definition 1. *[Blind Digital Signatures]. A blind digital signature scheme is a four-tuple* $\texttt{BlindSig} = (Signer, User, Gen, Verify)$ *where Gen and Verify are polynomial time algorithms, and*

- *Gen, the key generation algorithm, is a probabilistic algorithm that takes as an input an encoding of the security parameter k and outputs a pair* (pk, sk) *of public and secret keys.*
- *Verify, the verification algorithm, is a deterministic algorithm, which on input a triplet* (pk, m, σ) *outputs one bit meaning* accept/reject.

Signer and User are both interactive polynomially-bounded probabilistic Turing machines, each having the following (separate) tapes: read-only input tape, write-only output tape, a read/write work tape, a read-only random tape and two communication tapes, a read-only and a write-only tape. The User and Signer engage in an interactive protocol for some polynomial number of rounds. At this,

- *Signer takes as an input the key pair* (pk, sk)*, his output will be a single bit, meaning* completed/not-completed.
- *User takes as an input the public key pk together with a message m (of polynomial length in the security parameter). His output will be an error message* $\perp$ *or a signature* $\sigma(m)$.

[9]Basically Okamoto modified a previous definition by allowing the adversary to freely choose the public key and also to act dishonestly during `BlindSign` executions, without being forced to abort the game.

It must be the case that if both User and Signer follow the protocol specification, then Signer always outputs completed*, and the output* $\sigma(m)$ *User is always* accepted *by Verify; i.e.,* $Verify(pk, m, \sigma(m)) = 1$.

The following two properties must be achieved in order to consider a Blind Digital Signature scheme secure:

Definition 2. *[Non-forgeability Property]. Let* $\mathcal{A}$ *be a pptm adversary against a blind signature scheme* `BlindSig` *defined as above. Let* $\mathcal{C}$ *be a pptm challenger and consider the following game played by* $\mathcal{A}$ *and C :*

- $\mathcal{C}$ *runs the key generation algorithm* $\mathcal{K}$ *on input* 1^k *and retrieves a key pair* (pk, sk)*, and forwards the public key pk to* $\mathcal{A}$
- $\mathcal{A}$ *engages in L adaptive, parallel and arbitrarily interleaved interactive protocols with corresponding* $\mathcal{C}$ *acting as an honest Signer, all with input* (pk, sk)*. At this, L is decided adaptively by* $\mathcal{A}$*, but it is polynomial in k. Let l be the number of the above executions which* $\mathcal{C}$ *accepted as valid.*
- $\mathcal{A}$ *outputs a collection of j pairs* $(m_i, \sigma(m_i))$*, where all messages* m_i *in the list are different, and so that each pair is accepted by Verify on input pk.*

Then, `BlindSig` *is* non-forgeable *if for any probabilistic polynomial-time adversary* $\mathcal{A}$*, the probability, taken over coin-flips of Gen,* $\mathcal{A}$ *and* $\mathcal{C}$*, that* $j > l$ *is negligible in k.*

The above definition corresponds to the notion of security against "one-more" forgery considering parallel attacks from Pointcheval and Stern (see, for instance, (Pointcheval and Stern, 2000)).

Definition 3. *[Blindness Property]. Let* $\mathcal{A}$ *be a pptm adversary against a blind signature scheme* `BlindSig` *defined as above. Let* $\mathcal{S}$ *be a pttm challenger and consider the following game played by* $\mathcal{A}$ *and* $\mathcal{C}$

- $\mathcal{C}$ *generates the system parameters of the blind signature scheme which he forwards to* $\mathcal{A}$
- $\mathcal{A}$ *chooses a valid*[10] *public key,* pk_{BSig}*, and two different messages* m_0 *and* m_1 *to be signed, and sends all to* $\mathcal{C}$.
- *Now* $\mathcal{C}$ *flips a fair coin b and starts two instances of* `BlindSig` *on* m_b *and* m_{1-b}*, notifying* $\mathcal{A}$ *that the former is the target one.*

[10]Here 'valid' means one of the possible outputs of `IAKeyGen`.

- *At the end of the protocols, if $\mathcal{C}$ gets two valid blind signatures: σ_0 on m_0 and σ_1 on m_1, then $\mathcal{C}$ sends (σ_0, σ_1) to $\mathcal{C}$. Otherwise, if some of the protocols have been aborted or some of the signatures are not valid, $\mathcal{C}$ sends $\perp$ to $\mathcal{A}$.*
- *Finally, $\mathcal{A}$ ends the game by outputting a guess bit b'.*

Then the corresponding signature scheme fulfills the blindness property if the probability, taken over the choice of b, coin flips of Gen, $\mathcal{A}$ and $\mathcal{C}$ that $b = \hat{b}$ is bounded by

$$\frac{1}{2} + \varepsilon(k),$$

for some negligible function ε.