Secure Multi-Coupons for Federated Environments: Privacy-Preserving and Customer-Friendly

Frederik Armknecht[1], Alberto N. Escalante B.[1], Hans Löhr[1], Mark Manulis[2], and Ahmad-Reza Sadeghi[1]

[1] Horst Görtz Institute for IT Security
Ruhr-University of Bochum, Germany
{frederik.armknecht|alberto.escalante|
hans.loehr|ahmad.sadeghi}@trust.rub.de

[2] UCL Crypto Group, Université Catholique de Louvain, Belgium
mark.manulis@uclouvain.be

Abstract. A digital multi-coupon is similar to a paper-based booklet containing k coupons that can be purchased from one vendor and later redeemed at a vendor in exchange for services. Current schemes, offering privacy-protection and strong security properties such as unsplittability of multi-coupons, address business scenarios with a single vendor and multiple customers, and require customers to redeem coupons in some fixed order.

In this paper, we propose a multi-coupon scheme for federated environments that preserves the security and privacy properties of existing schemes, as well as their asymptotic communication and computation complexity. We define a generic formal security model and show that our scheme meets the formal requirements of this framework. Moreover, in contrast to previous solutions, we allow customers to redeem their coupons in an arbitrary order.

Keywords: coupons, privacy, unlinkability, unsplittability, payment system, loyalty, federation

1 Introduction

Coupons are the basis for successful business models and are widely used in practice. Companies distribute (paper-based) coupons to customers for various marketing purposes, like encouraging loyalty, providing discounts, setting up prepayment models, and attracting new customers. A special variant are coupon booklets, where all coupons are contained in a booklet and are only valid as long as they are attached to the booklet. This ensures a property we call *unsplittability*: the single coupons cannot be redeemed autonomously; instead, they can only be shared among customers by giving away the entire booklet each time a coupon is spent.

We call coupon booklets (and their electronic equivalents) *multi-coupons (MCs)*. A vendor provides a customer with a new multi-coupon in the *issue* procedure. The customer can then use the coupons from this multi-coupon in the *redeem* procedure to pay the vendor. During redemption, the vendor verifies that the coupon is valid and authentic, and provides the customer with the specified good or service. Each coupon in a multi-coupon can be used only once. In the following, we denote by *object* the good or service implied by a coupon. Any item that can be bought may become an object in practice, e.g., clothes, songs, books, videos, tickets, and even services, such as discounts, access to computer resources, etc.

Multi-Coupons in a Vendor Federation. Until now, multi-coupons were proposed in use cases with a single vendor. Hence, one approach to make MCs more user-friendly is to make them usable in a more general scenario, where a federation of vendors is involved. For instance, consider a cooperation between transportation companies, different cultural institutions, restaurants, and shops offering joint coupons to tourists who can then visit any of the indicated places of interest, eat at the participating restaurants, and buy goods from the listed shops at discount prices. Note that paper-based variants of such cooperations exist in many cities (e.g., [1–3]) and enjoy popularity. The general case is that tourists buy a special card which is accompanied with coupons offering discounts. This card should be presented prior to using any of the coupons (i.e., coupons are unsplittable).

Obviously, it would be more convenient if a tourist could buy the MC at *any* involved vendor and would not be forced to go to a central place like the tourist information. A trivial solution could be to connect each vendor to one central server which issues the electronic MCs, but more intelligent solutions which allow the vendors to act autonomously as much as possible are certainly preferrable. Moreover, there might be several competing vendors in the federation that provide the same service, e.g., different restaurants, where a customer could get a meal at a reduced price. In such cases, it might be desirable that the vendor who actually provided the service – e.g., the restaurant which served the meal – obtains money for it. In our scheme, a vendor can prove that a customer redeemed a coupon to him, and hence he could charge the coupon issuer.

We remark that the above scenario is just a use case where a digital multi-coupon scheme maintained by a federation of vendors would be of potential interest, and that the scheme designed in this paper is general and could be employed in different business models through the specification of its object types.

Electronic *multi-coupon schemes (MCSs)* are in several ways superior to paper-based schemes. Despite the lower production costs and the possibility to buy and generate them over the Internet, they enable finer business models tailored to the different types of customers. However, new and specific security considerations need to be taken into account.

Security and Privacy Considerations. In contrast to paper-based coupon booklets, it is very easy to create a perfect copy of an electronic MC. Further, when dealing with an MCS, we must also consider attacks in which different users collude and attempt to cheat on vendors. Moreover, privacy and anonymity of customers become more important since the vendor may try to infer and store additional information about them including purchase habits, gender, age, etc. This would harm privacy and allow client profiling and price discrimination [17]. For optimal user privacy, vendors should not be able to link different transactions to one user (i.e., *unlinkability* should be provided).

Unforgeability and *unsplittability* are essential properties of an MCS (see [8, 10, 15]). The users should not be able to forge coupons or share (“split”) an MC in such a way that several users can spend coupons from one MC independently. In the literature, *weak unsplittability* (also called *all-or-nothing sharing*) has been proposed (see, e.g., [10]): a user who wants to share a single coupon with someone else has to share the entire MC and all the secret data associated with that MC. Our scheme fulfills a stronger definition, called *unsplittability* (cf. [11]): If two users share coupons from an MC, then if one of them redeems, the second one cannot redeem any coupon from the same MC without interaction with the first user, even if both users know the entire secret data. To support business models where the vendor which provides the user with a service can charge money from the issuer of the coupon, additional requirements must be met. During the redemption protocol, the issuer of the coupon must be identifiable, and other vendors must be protected from being incorrectly held responsible for issuing this coupon. In Section 3, we actually define two requirements, *framing resistance* (the requirement of the issuer) and *claimability* (the requirement of the redeeming vendor).

Although payment issues are important for the deployment of an MCS in practice, they cannot be completely solved by cryptographic techniques. Hence, these issues are out of scope of this paper. Here, we assume that it suffices that a judge can execute an algorithm `Claim` to verify that a coupon, issued by a given issuer, has been redeemed to a given vendor.

Contribution. We introduce a new multi-coupon scheme deployable for a federation of vendors. Our scheme provides unlinkability, unsplittability, unforgeability, framing resistance and claimability. We introduce a formal security framework with definitions of these properties in which we prove the security of our scheme.

Previous MCSs suffer from the problem that they either do not provide unsplittability, or all the coupons in a multi-coupon have to be redeemed in sequential order (fixed during issue). If an MCS is to be used with a federation of vendors, such a restriction can be a strong limitation: imagine that the vendors want to offer an MC with coupons for different types of goods. In that case, customers certainly would want to decide themselves in which order they want to redeem their coupons. Hence, we need a *non-sequential* MCS, where the coupons can be redeemed in arbitrary order. However, the scheme of [11] offers nice features that we want to retain, in particular, *coupon objects*. These allow to have different types of coupons in one MC. We improve and extend this scheme in two important aspects: our scheme can be used by a group of vendors, which also introduces new security requirements. Moreover, we do not require the order of redemption of the single coupons to be fixed when the MC is issued. Furthermore, MCs can be created and issued offline without any connection to the vendors at which the coupons can be redeemed. For instance, this allows in practice to install a variety of selling booths in the tourist card example mentioned above.

Redeem complexity (both computation and communication) is constant w.r.t. the size k of the MC (i.e., the number of coupons it contains), and complexity of the protocol for issuing MCs is linear in k, which is the best we can get when each coupon has individual attributes (like coupon objects). If all coupons in an MC are the same (i.e., no coupon objects are used), ideas from [6] can be used to further reduce the complexity.

Organization. First, we give an overview of our scheme, define general multi-coupon schemes and describe our realization in Section 2. In Section 3, we give a formal framework with game-based formal definitions of the requirements, and provide sketches for security proofs. We discuss related work in Section 4. Finally, we conclude our article in Section 5. Further details can be found in the extended version.[3]

2 Our Federated Multi-Coupon Scheme

2.1 Informal Description of a Multi-Coupon's Lifecycle

In our scheme, a group of vendors $\mathcal{V}$ with common databases DB, DB' (trusted by the vendors) executes protocols with users $\mathcal{U}$ to issue and redeem coupons. The databases are used only during the redeem protocol. A multi-coupon M contains k individual coupons, which include, among other information, a coupon identifier id. The coupons are all cryptographically tied to M, which has an MC identifier mid and a freshness

[3] see `http://www.trust.rub.de/home/publications`

identifier *fid*. To simplify the description below, we temporarily omit coupon objects ob and the MC identifier mid.

In the `Issue` protocol, a user U obtains an MC from a vendor V with one signature on each individual coupon, and one signature validating the freshness *fid*, signed by the issuing vendor V. The signatures on the individual coupons (on id) prevent U from forging coupons, whereas the signature on the MC (on *fid*) ensures its freshness, which is used to prevent splitting.

In the `Redeem` protocol, the user U redeems a single coupon from an MC to a vendor V'. For this, he has to prove knowledge of a signature on the single coupon and that the MC is fresh. Double redemption of coupons is prevented by the vendor V' through a lookup in a central database DB of coupon identifiers. Similarly, V' queries the central database DB' of freshness IDs to verify the freshness of the MC. If the current coupon id and freshness ID *fid* have not already been used, then they are inserted into the corresponding database. Afterwards, the database DB sends a signature $\texttt{cert}_{DB}$ to the vendor V' certifying that V' is responsible for the redemption of this coupon. V' will need this signature as an evidence to charge the coupon issuer. At the end of `Redeem`, a new *fid* is generated and signed by V', so that this protocol can be executed repeatedly, as long as there are coupons left in the MC.

After redemption, the `Claim` algorithm can be executed by any party to verify that a user redeemed a coupon originally issued by a vendor V to a vendor V', and thus, that V' is entitled to charge V for the corresponding coupon. The input to this algorithm is the coupon ID id, a (non-interactive) proof of knowledge of a signature on id, and the certificate $\texttt{cert}_{DB}$ given by DB to V' during `Redeem`. The certificate is used to prevent double charging. Note that the databases do not participate in this algorithm.

2.2 Components of a General Federated MCS

Basic Notation. For a finite set $\mathcal{S}$, $s \in_R \mathcal{S}$ denotes the assignment of an element sampled uniformly from $\mathcal{S}$ to the variable s. Let Alg_A be a probabilistic algorithm. By $\mathrm{out}_A \leftarrow \mathrm{Alg}_A(\mathrm{in}_A)$ we denote that the variable out_A is assigned the output of Alg_A's execution on input in_A. We denote by $(\mathrm{Alg}_A(\mathrm{in}_A), \mathrm{Alg}_B(\mathrm{in}_B))$ a pair of interactive algorithms with private inputs in_A and in_B, respectively, and write $(\mathrm{out}_A, \mathrm{out}_B) \leftarrow (\mathrm{Alg}_A(\mathrm{in}_A), \mathrm{Alg}_B(\mathrm{in}_B))$ to denote the assignment of Alg_A's and Alg_B's private outputs after their interaction to the variables out_A and out_B, respectively.

Here, we adapt the basic framework from [11] to our scenario with a federation of vendors. The involved parties are a set of vendors $\mathcal{V}$ and a set of users $\mathcal{U}$, where $n_{\mathcal{V}} = |\mathcal{V}|$ denotes the number of vendors in the federation. We will refer to any particular user simply by U, and V, V' will denote particular vendors. We assume that each vendor V has a unique identity ID_V which is publicly known. Common system parameters for the cryptographic building blocks (like commitment and signature schemes) will be omitted in the notation for better readability.

Definition 1 (Multi-Coupon Scheme). *A* multi-coupon scheme *(MCS) for a federation of vendors $\mathcal{V}$ consists of a set of protocols and algorithms {*`Setup`*,* `Issue`*,* `Redeem`*,* `Claim`*}:*

Setup **algorithm**. $(PK, \{SK_{V_i}\}_{1 \le i \le n_{\mathcal{V}}}) \leftarrow \mathtt{Setup}(1^\kappa, n_{\mathcal{V}})$ *is the (in general, distributed) initialization algorithm executed by the vendors once to generate one instance of the MCS, where κ is the security parameter, $n_{\mathcal{V}}$ is the number of vendors. It outputs a public key PK (which includes 1^κ and $k_{\max}$, the maximum allowed number* *of coupons per MC), and a set of secret keys $\{SK_{V_i}\}_{1\le i\le n_V}$.The vendors' states are initialized to the empty string.*

***Issue* protocol**. *In order to obtain an MC with k coupons, U performs the following protocol with a vendor V: $((res_u, M), res_v) \leftarrow (\texttt{Issue}_u(k, V, PK, ob_0, \ldots, ob_{k-1}), \texttt{Issue}_v(k, SK_V, ob_0, \ldots, ob_{k-1}))$ where, from now on, the subindices u and v denote user and vendor algorithms, respectively. The common input $ob_0, \ldots, ob_{k-1}$ specifies coupon objects (individual attributes) for the k individual coupons in the MC that is to be issued. The output flags res_u, $res_v \in \{acc, rej\}$ indicate success or failure. $\texttt{Issue}_u$ outputs res_u and a multi-coupon M, whereas $\texttt{Issue}_v$ only outputs res_v.*

***Redeem* protocol**. *A multi-coupon M (issued by V) is redeemed to V' via the protocol $((res_u, M'), (res_v, crn, ob, \pi, s')) \leftarrow (\texttt{Redeem}_u(M, m, PK), \texttt{Redeem}_v(s, SK_{V'}))$. The parameters to $\texttt{Redeem}_u$ are the multi-coupon M from which the user wants to redeem a coupon, a specification m of the coupon to be redeemed[4], and the public key PK of the MCS. The vendor algorithm takes the vendors' state s and the private key of the redeeming vendor $SK_{V'}$ as input. $\texttt{Redeem}_u$ outputs an updated multi-coupon M' and a flag res_u just like in $\texttt{Issue}$, and $\texttt{Redeem}_v$ outputs a new state s' of the vendors, a unique coupon reference number crn, an object ob, a proof π that a user redeemed a coupon to V' (with reference number crn and object ob, issued by V), and a flag res_v.*

***Claim* algorithm**. *To verify that a coupon with reference number crn issued by V has indeed be redeemed to vendor V', the (public) algorithm $\texttt{Claim}$ can be run to verify a proof π, i.e., res $\leftarrow \texttt{Claim}(crn, ob, \pi, V', V)$. The result res is $\texttt{true}$ if π proves that V issued a coupon with object ob that was redeemed to V' with reference number crn; otherwise, res is $false$. crn is used to identify a redeemed coupon, i.e., it can be noticed, when the same redeemed coupon is claimed twice.*

***Correctness** (informal).* Any MCS must fulfill the correctness requirement: if all participants in the protocol are honest, each individual coupon from each MC that was issued by any vendor can be redeemed successfully at any vendor (equal to or different from the issuer), regardless of the order of redemption, i.e., a user can redeem any coupon that she hasn't spent yet at any time.

2.3 Building Blocks

Commitment Scheme (CS). We use the integer CS from [7], based on the scheme in [12], with two bases $g, h \in \mathrm{QR}_n$ (quadratic residues modulo n), and a special RSA modulus n as a public key. A commitment to x has the form $C_x = g^x \cdot h^r$, where r is a random value.

Proofs of Knowledge (PoK). We use a number of honest-verifier statistical zero-knowledge PoKs. By $\texttt{PoK}\{(\tilde{x}_1, \ldots, \tilde{x}_n) : R(\tilde{x}_1, \ldots, \tilde{x}_n)\}$ we denote an interactive PoK, where a prover proves to a verifier that she knows a witness $(\tilde{x}_1, \ldots, \tilde{x}_n)$ (denoted by tilded variables) such that relation R holds, and the verifier does not gain any useful information beyond this assertion.

Proof of Equality of Representations. $\mathcal{P}$ proves that she is able to open two commitments C_1 and C_2 (for two possibly different instances of the commitment scheme), such that certain components of the openings are equal. For example, $\texttt{PoKEqRep}\{(\tilde{x}, \tilde{r}_x,$

[4] Details depend on the scheme; e.g., m could be the index in a list of all coupons in a multi-coupon or an ID.

$\tilde{y}, \tilde{r}_y) : C_1 = g_1^{\tilde{x}} g_2^{\tilde{r}_x} \wedge C_2 = \hat{g}_1^{\tilde{y}} \hat{g}_2^{\tilde{r}_y} \wedge \tilde{x} = \tilde{y}\}$ denotes the proof that the exponents $\tilde{x}$ and $\tilde{y}$ are equal.

Camenisch Lysyanskaya signature scheme (*CLS*). The *CLS* [7] is a signature scheme with efficient protocols based on the *strong RSA assumption*. The protocols for this scheme allow signing committed values, and proving knowledge of a signature (see below). The following description is done in the context of our scheme.

$\mathit{CLS}.\mathtt{Setup}(1^\kappa)$. The signer $\mathcal{S}$ generates a special RSA modulus $n = pq$, such that n has size $\ell_n := 2\kappa$, where κ is a security parameter. Then he chooses numbers $a, b, c \in_R \mathrm{QR}_n$, where a, b are called bases. The public key CLS_{PK} is (a, b, c, n), and the secret key CLS_{SK} is the prime p.

$\mathit{CLS}.\mathtt{Sign}(x, \mathit{CLS}_{SK})$. To sign a message $x \in [0; 2^{\ell_m})$, the signer chooses a random prime e of size $\ell_e := \ell_m + 2$, a random number s of size at most $\ell_s := \ell_n + \ell_m + \ell$, where ℓ is another security parameter, $\mathcal{S}$ computes $v \leftarrow (a^x b^s c)^{e^{-1}} \pmod{n}$, and outputs (e, s, v).

$\mathit{CLS}.\mathtt{Verify}(x, \sigma, \mathit{CLS}_{PK})$. For $(e, s, v) := \sigma$, the algorithm tests if $v^e \equiv a^x b^s c \pmod{n}$, $x \in [0; 2^{\ell_m})$, $s \in [0; 2^{\ell_s})$, e is ℓ_e bits long, and outputs *true* or *false*.

The signature allows the following useful protocols:

Signature on a committed value and PoK of this signature [7]. Signature generation is a protocol from [7] between a user U and a signer $\mathcal{S}$, who knows the secret key CLS_{SK}. Let $\mathit{CLS}_{PK} := (a, b, c, n)$ be the corresponding public key. The common input to U and $\mathcal{S}$ is a commitment C_x, for which U (supposedly) knows an opening $(x, r_x) : C_x = a^x b^{r_x}$. At the end of the protocol U obtains a signature $\sigma := (e, s, v)$ on x, while x is statistically hidden from $\mathcal{S}$. We denote this protocol as: $\sigma \leftarrow \mathtt{SigOnCommit}\{U(x, r_x), \mathcal{S}(\mathit{CLS}_{SK})\}(C_x)$.

For a commitment C'_x, U can prove knowledge of (x, r'_x, e, s, v) [7], such that (x, r'_x) is an opening of C'_x, and (e, s, v) is a valid signature on x, where x and σ are hidden by the zero-knowledge property of the protocol. We denote this protocol as: $\mathtt{PoKSigOnCommit}\{(\tilde{x}, \tilde{r}'_x, \tilde{\sigma}) : C'_x = a^{\tilde{x}} b^{\tilde{r}'_x} \wedge \mathit{CLS}.\mathtt{Verify}(\tilde{x}, \tilde{\sigma}, \mathit{CLS}_{PK})\}$.

This signature scheme can be extended to sign message tuples $(x_1, \ldots, x_k)$ by introducing k bases a_i [7]. The extended scheme for k-tuples will be denoted by *CLSk*. The protocols above can be extended to support multiple messages, and selective message disclosure. E.g., abusing notation, we denote by $\mathtt{SigOnCommit}\{\ U(\tilde{x}_1, \tilde{r}_{x_1}), \mathcal{S}(\mathit{CLS3}_{SK})\}(C_{x_1}, x_2, x_3)$ a protocol to generate a signature on a 3-tuple (x_1, x_2, x_3), where the message x_1 is blinded by a commitment C_{x_1}, and two messages x_2 and x_3 are disclosed in clear. Similarly, by $\mathtt{PoKSigOnCommit}\{(\tilde{x}_3, \tilde{r}_{x_3}, \tilde{\sigma}) : C_{x_3} = a_3^{\tilde{x}_3} b^{\tilde{r}_{x_3}} \wedge \mathit{CLS3}.\mathtt{Verify}((x_1, x_2, \tilde{x}_3), \tilde{\sigma}, \mathit{CLS3}_{PK})\}$ we denote the corresponding PoK that U knows a signature σ on a tuple (x_1, x_2, x_3), where x_1 and x_2 are disclosed to the verifier, but x_3 is kept blinded. Again, the variables with ˜ are kept secret.

Non-interactive proofs and signatures of knowledge. Using a cryptographic hash function, the PoKs described above can be turned into non-interactive PoKs by the Fiat-Shamir heuristic [13]. We add the prefix `NI-` ("non-interactive") to the PoKs to indicate that a non-interactive proof is used instead of an interactive protocol, e.g., `NI-PoKSigOnCommit` to denote a non-interactive proof of knowledge of a signature on a commitment. If additional data (a "message") is hashed, the NI-PoK becomes a signature on this message (as in [19]) and is called a *signature of knowledge (SoK)*. Since the actual protocol remains the same, we use the same notation with simply appending the message (as in $\mathtt{NI\text{-}PoKSigOnCommit}\{\ldots\}(m)$). The security of SoKs can be shown in the *random oracle model*. In practice, it is assumed that this heuristic is secure, as long as the hash function which is used is cryptographically strong. For a more general and formal treatment of SoKs, see [9].

2.4 Concrete Construction

Overview. A multi-coupon M of size $k \leq k_{\max}$ consists of its identifier mid, a freshness identifier fid, a signature σ' on the pair (fid, mid), and a list of k individual coupons, where $k_{\max}$ is the maximal number of coupons an MC can contain. Each individual coupon (id, ob, σ) is specified by a coupon identifier id, a coupon's object ob (i.e., the good or service represented by the coupon[5]), and a signature σ on the tuple (id, ob, mid). Depending on the business model, the object IDs in an MC could either be chosen by the user, or they could be determined by the issuer. We model object IDs as common input to the issue protocol, leaving this decision to the concrete application.

We require that all signatures and non-interactive proofs in the protocols are always verified by the recipient. If the verification fails, the protocol is aborted, and the respective party outputs *rej* (subsequently, verification steps will be omitted). All public keys and parameters for the underlying protocols are known to all participants in the scheme (e.g., the federation of vendors could maintain a server with a directory of all public keys). The coupon reference number crn from our formal definitions is implemented by a unique ID id_i for each individual coupon.

Setup. For the setup of the MCS, the vendors have to create keys[6]: one common $\mathit{CLS2}$ key pair (PK_{Fed}, SK_{Fed}) for the federation, where all vendors know the private key, and one $\mathit{CLS3}$ key pair (PK_V, SK_V) for each individual vendor V. Moreover, the vendors have to create two empty common databases DB (for coupon IDs) and DB' (for freshness IDs), where all vendors can create new entries (of course, this can be implemented by two tables in one database). Every vendor is allowed to insert entries into the databases, but no vendor is allowed to delete them. DB possesses a key pair (PK_{DB}, SK_{DB}) of an arbitrary signature scheme, e.g., RSA, to issue certificates to vendors which inserted coupon IDs.

Remark. In this instantiation, the public key mentioned in Def. 1 consists of PK_{Fed} and PK_{V_i}; the secret key from Def. 1 includes SK_{Fed} and SK_{V_i}.

Issue. The `Issue` protocol is shown in Fig. 1. In step 1, the multi-coupon identifier mid is selected by the vendor, whereas the freshness ID fid_0 and IDs for the individual coupons id_i are chosen by the user. The vendor only obtains commitments $C_{\mathit{fid}_0}, C_{id_0}, \ldots C_{id_{k-1}}$ to the values chosen by the user. In step 2, the user receives a signature σ_0' on (mid, fid_0) with the secret key of the federation SK_{Fed}, and in step 3, he obtains signatures σ_i on (C_{id_i}, mid, ob_i) with the signing key SK_V of the issuer.

Redeem. The `Redeem` protocol for the $(j + 1)$-th redemption from a multi-coupon, where $0 \leq j \leq k-1$, is shown in Fig. 2. During the first `Redeem` from a multi-coupon (i.e., $j = 0$), the freshness ID fid_0 and corresponding signature σ_0' from `Issue` is used and updated; in subsequent redemptions, the freshness ID and signature from the previous execution of `Redeem` are used and updated. In step 1, the user blinds mid by commitments (otherwise, the vendor could use mid to link transactions), and sends the data of the coupon he wants to redeem $(id_i, ob_i, \mathit{fid}_j)$, together with the ID of the issuer ID_V, to the vendor V'. In step 2, U proves that the two commitments to mid are actually commitments to the same number. In step 3, the user proves knowledge of the

[5] The vendors must publish an encoding of coupon's objects as integers.

[6] We do not use group signatures, because coupon issuers should be identifiable.

Common input: public keys $PK_V = (a_1, a_2, a_3, b, c, n)$, $PK_{Fed} = (\hat{a}_1, \hat{a}_2, \hat{b}, \hat{c}, \hat{n})$,
number of single coupons k, object identifiers ob_i, $i = 0, \dots, k-1$
User's input: $-$
Vendor's input: private keys $SK_V = p$, $SK_{Fed} = \hat{p}$

```
User U                                                                Vendor V
Step 1:
  $fid_0 \in_R (0; 2^{\ell_m})$; $r_{fid_0} \in_R (0; 2^{\ell_n})$;                    $mid \in_R (0; 2^{\ell_m})$;
  $C_{fid_0} \leftarrow \hat{a}_1^{fid_0} \hat{b}^{r_{fid_0}}$;          $\xleftarrow{\quad mid \quad}$
  for each $i = 0, \dots, k-1$ do
    $id_i \in_R (0; 2^{\ell_m})$;
    $r_{id_i} \in_R (0; 2^{\ell_n})$;
    $C_{id_i} \leftarrow a_1^{id_i} b^{r_{id_i}}$;            $\xrightarrow{C_{fid_0}, C_{id_0}, \dots, C_{id_{k-1}}}$
  end for;
Step 2:
  $\sigma'_0$   $\leftarrow$   $\texttt{SigOnCommit}\{U(fid_0, r_{fid_0}), V(SK_{Fed})\}(C_{fid_0}, mid)$
                                  $\longleftrightarrow$
Step 3:
  for each $i = 0, \dots, k-1$ do
    $\sigma_i$   $\leftarrow$   $\texttt{SigOnCommit}\{U(id_i, r_{id_i}), V(SK_V)\}(C_{id_i}, mid, ob_i)$
                                  $\longleftrightarrow$
  end for;
  return $(mid, fid_0, \sigma'_0, \{(id_i, \sigma_i)\}_{0 \le i < k})$;                return $accept$;
```

Fig. 1. `Issue` Protocol.

signature σ_i, and the vendor obtains a signature of knowledge π' that allows him later to prove that this coupon was redeemed to him. In step 4, the user proves knowledge of a signature σ'_j on (fid_j, mid). The vendor has to verify that both id_i and fid_j are fresh by quering the databases (i.e., he checks that these values are not yet in DB and DB'), and inserts these entries. After insertion, the database DB signs id_i and sends the signature to V'. To prevent races between vendors, which open the door to some attacks, only one vendor at any time is allowed to "query and insert", as an atomic operation.

In step 5, U chooses a new random freshness ID fid_{j+1} for this MC and sends a commitment to fid_{j+1} to V'. At the end of the protocol (in step 6), the user obtains a new freshness signature σ'_{j+1} for this MC. The vendor sets $\pi \leftarrow (\pi', \texttt{cert}_{DB}, C_{mid})$, and returns (id_i, ob_i, π).

A malicious user cannot abuse $C_{fid_{j+1}}$ to obtain signatures with SK_{Fed} on arbitrary messages, because the second part of the signed message is proven to be a valid commitment to mid. All signatures with SK_{Fed} on such messages will always be interpreted as freshness signatures, thus this protocol cannot be used as signature oracle. For efficiency reasons, the NI-PoKs and NI-SoKs could all be combined into one NI-SoK.

Claim. The deterministic `Claim` algorithm verifies the SoK that a vendor V' obtained during the `Redeem` protocol and the certificate given by DB to V'. It uses only public information and hence can be run by anyone, for example, by a judge in case of dispute. Double charging is prevented because a vendor will only pay back once for each coupon identifier. The vendor V' can always charge the issuing vendor unless DB generates two certificates for the same coupon identifier. However, this misbehavior can always be identified.

```
Common input: public keys $PK_V = (a_1, a_2, a_3, b, c, n)$, $PK_{Fed} = (\hat{a}_1, \hat{a}_2, \hat{b}, \hat{c}, \hat{n})$
    bases $g, h, \hat{g}, \hat{h}$ for internal use of PoKSigOnCommit protocols
User's input: single coupon $(id_i, mid, ob_i, fid_j, \sigma_i, \sigma'_j)$ issued by vendor $V$, issuer's ID $ID_V$
Vendor's input: private key $SK_{Fed} = \hat{p}$, databases $DB$, $DB'$
        User $U$                                                        Vendor $V'$
Step 1:
    $r_{mid}, r'_{mid} \in_R (0; 2^{\ell_n})$;
    $C_{mid} \leftarrow a_2^{mid} b^{r_{mid}}$;
    $C'_{mid} \leftarrow \hat{a}_2^{mid} \hat{b}^{r'_{mid}}$;          $\xrightarrow{ID_V, id_i, C_{mid}, ob_i, fid_j, C'_{mid}}$
Step 2:
NI-PoKEqRep$\{(\tilde{mid}, \tilde{mid}', \tilde{r}_{mid}, \tilde{r}'_{mid}) : C_{mid} = a_2^{\tilde{mid}} b^{\tilde{r}_{mid}} \wedge C'_{mid} = \hat{a}_2^{\tilde{mid}'} \hat{b}^{\tilde{r}'_{mid}} \wedge \tilde{mid} = \tilde{mid}'\}$ $\longrightarrow$
Step 3:
NI-PoKSigOnCommit$\{(\tilde{mid}, \tilde{r}_{mid}, \tilde{\sigma}_i) : C_{mid} = a_2^{\tilde{mid}} b^{\tilde{r}_{mid}} \wedge CLS3.$Verify$((id_i, \tilde{mid}, ob_i), \tilde{\sigma}_i, PK_V)\}(ID_{V'})$ $\longrightarrow$
                                                        $\pi' \leftarrow$ output of step 3;
Step 4:
NI-PoKSigOnCommit$\{(\tilde{mid}, \tilde{r}'_{mid}, \tilde{\sigma}'_j) : C'_{mid} = \hat{a}_2^{\tilde{mid}} \hat{b}^{\tilde{r}'_{mid}} \wedge CLS2.$Verify$((fid_j, \tilde{mid}), \tilde{\sigma}'_j, PK_{Fed})\}$ $\longrightarrow$
Step 5:                                                 cert$_{DB} \leftarrow$ insert $id_i$ into $DB$;
    $fid_{j+1} \in_R (0; 2^{\ell_m})$;  $r_{fid_{j+1}} \in_R (0; 2^{\ell_n})$;      insert $fid_j$ into $DB'$;
    $C_{fid_{j+1}} \leftarrow \hat{a}_1^{fid_{j+1}} \hat{b}^{r_{fid_{j+1}}}$;          $\xrightarrow{C_{fid_{j+1}}}$
Step 6:
    $\sigma'_{j+1}$   $\leftarrow$   SigOnCommit$\{U(fid_{j+1}, r_{fid_{j+1}}, mid, r'_{mid}), V'(SK_{Fed})\}(C_{fid_{j+1}}, C'_{mid})$ $\longleftrightarrow$

    return $(fid_{j+1}, \sigma'_{j+1})$;                $\pi \leftarrow (\pi', $cert$_{DB}, C_{mid})$;   return $(id_i, ob_i, \pi)$;
```

Fig. 2. Redeem Protocol.

```
Claim$(id, ob, \pi, V', V)$ :
    parse $\pi$ as $(\pi', $cert$_{DB}, C_{mid})$;
    verify cert$_{DB}$ w.r.t. $id, PK_{DB}$;
    verify $\pi'$ w.r.t. $id, ob, C_{mid}, PK_V, ID_{V'}$;
```

Fig. 3. Claim Algorithm.

Efficiency. The communication (and computation) complexity of the Issue protocol is linear in the number k of individual coupons in the multi-coupon to be issued. Correspondingly, the size of the MC data is also linear in k. The Redeem protocol is constant w.r.t. to k. The operations performed by DB and DB' (search, insert and sign) do not depend on the size k of the MCs (but, of course, on the security parameter κ), and they should not impact the efficiency unless the communication between the vendors and the databases is slow. If coupon objects are not necessary, ideas from [6] could be used to obtain logarithmic complexity (in k) for Issue, and also logarithmic size of the MC data. Compared to the MCS from [11], one additional SigOnCommit protocol has to be run instead of a local signature generation during Issue. In the Redeem protocol, two additional IDs (V and fid_j) are sent to the vendor in the first step, and we need an extra round to send a commitment to the vendor. Another difference is that we use non-interactive versions of the protocols during Redeem, which slightly increases efficiency – but this could also be done in the MCS from [11].

3 Security Framework and Analysis

Here, we generalize the adversarial model from [11] to a federation of vendors. The security requirements are defined by *games*, and it can be shown that our scheme meets these requirements. We only present some proof sketches and refer the reader to the full version of this paper for more details. An adversary is a p.p.t. algorithm $\mathcal{A}$, which can play the role of either a collusion of vendors and users, or only of a group of users. W.l.o.g., we let the adversary be specified by a sequence of algorithms (e.g., $\mathcal{A} := (A_1, A_2, A_3)$). Honest parties are assumed to communicate over secure channels.

We consider two types of users (resp. vendors): *honest* and *corrupted* users (resp. vendors). Users (resp. vendors) belonging to the set of honest users (resp. vendors) execute algorithms of the MCS if requested by $\mathcal{A}$, but remain honest otherwise. $\mathcal{A}$ has full control over the corrupted users and vendors, and he is provided with their previous protocol views. Similar to [14], we allow $\mathcal{A}$ to interact with the system through a set of queries[7] handled by an *interface*, which partially simulates the MCS, executes protocols with $\mathcal{A}$, and records certain user's or vendor's activities. Note that the interfaces do not restrict $\mathcal{A}$ in any way – they control the actions of the honest parties on behalf of $\mathcal{A}$. Correctness of the scheme can be easily verified (proof omitted).

Framing resistance and claimability. During the redemption protocol, the original issuer of the coupon must be identifiable (to allow the redeeming vendor to claim money from the issuer), and other vendors must be protected from false claims. It must be ensured that a vendor who issued an MC can always be held responsible for all coupons from this MC. We break down this property into two requirements: (1) **framing resistance**: a collusion of vendors and users must never be able to claim that another vendor issued a coupon with a specific object, when he didn't; and (2) **claimability**: an honest vendor who redeemed a coupon must always be able to claim money for it.

Interface I_1. In the games defining "claimability" and "framing resistance", the adversary $\mathcal{A}$ plays the role of a coalition of all users and has the capability to corrupt vendors.

Counters $ctrC_{V,ob}$ (initially 0) for each coupon object ob are defined for each vendor V, counting the coupons with object ob, that were issued by V. The following queries are provided to $\mathcal{A}$.

$I_1.\texttt{Issue}_v(V, k, ob_0, \ldots, ob_{k-1})$. If $k \in [1; k_{\max}]$ and V is an honest vendor, the $\texttt{Issue}_v$ algorithm is executed. The counter for each coupon object ob is increased by the number of times ob occurs in the MC issued by V, i.e., $\forall \lambda \in [0; k-1]$: $ctrC_{V,ob_\lambda}$++.

$I_1.\texttt{Redeem}_v(V', V)$. If V' is an honest vendor, the $\texttt{Redeem}_v$ protocol is executed for V', i.e., $\mathcal{A}$ wants to redeem a coupon (issued by V) to V'.

$I_1.\texttt{Corrupt}(V)$. $\mathcal{A}$ receives all secrets of V (and V is removed from the set of honest vendors).

In the $FrameGame$ (see Fig. 4), $\mathcal{A}$ can interact with the system via the interface I_1. $\mathcal{A}$ outputs the identity V of the vendor he wants to "frame" (in order to win this game, $\mathcal{A}$ has to choose an uncorrupted vendor), an object ob, and a set of coupon reference numbers CRN with a corresponding set Π of pairs (π, V') of proofs that V' was involved in the redemption of a coupon with object ob issued by V. If $\texttt{Claim}$ succeeds for all of these proofs and there are more elements in CRN than coupons (with object

[7] Like in existing schemes, queries must not be executed concurrently, which simplifies model and construction.

ob) issued by V (i.e., $|CRN| > ctrC_{V,ob}$), $\mathcal{A}$ wins the game, because then $\mathcal{A}$ must be able to claim coupons V did not issue. (Of course, all elements of the set must be distinct – i.e., $\mathcal{A}$ cannot "replay" the same crn multiple times).

```
$FrameGame(\mathcal{A}, \kappa, n_{\mathcal{V}})$:
  $(PK, \{SK_{V_i}\}_{1 \leq i \leq n_{\mathcal{V}}}) \leftarrow$
      Setup$(1^\kappa, n_{\mathcal{V}})$;
  $(V, ob, CRN, \Pi) \leftarrow \mathcal{A}^{I_1}(1^\kappa, PK)$;
  if $\Big($ $V$ uncorrupted $\wedge\ |CRN| > ctrC_{V,ob} \wedge$
     $(\forall crn \in CRN\colon \exists(\pi, V') \in \Pi$:
       Claim$(crn, ob, \pi, V', V) = true)\Big)$
        return broken;
  else return unbroken;
```

```
$ClaimGame(\mathcal{A}, \kappa, n_{\mathcal{V}})$:
  $(PK, \{SK_{V_i}\}_{1 \leq i \leq n_{\mathcal{V}}}) \leftarrow$
      Setup$(1^\kappa, n_{\mathcal{V}})$;
  $(V', V, s_{\mathcal{A}}) \leftarrow A_1^{I_1}(1^\kappa, PK)$;
  if $V'$ corrupted
      return unbroken;
  $(Res_{\mathcal{A}}, (res_v, crn, ob, \pi, s')) \leftarrow$
      $(A_2(s_{\mathcal{A}}),\ I_1$.Redeem$_v(V', V))$;
  if $(res_v = acc\ \wedge$
     Claim$(crn, ob, \pi, V', V) = false)$
      return broken;
  else return unbroken;
```

Fig. 4. The games $FrameGame$ and $ClaimGame$.

Definition 2 (Framing resistance of an MCS). *An MCS is* resistant against framing *if there is no p.p.t adversary $\mathcal{A}$ that can win the $FrameGame$ in Fig. 4 (i.e., Frame-Game$(\mathcal{A}, \kappa, n_{\mathcal{V}}) =$ broken for some number of vendors $n_{\mathcal{V}} \geq 1$) with non-negligible probability (in κ).*

Theorem 1 (Framing resistance). *Assuming the security of CL signatures against existential forgery, the proposed MCS is resistant against framing, i.e., for all p.p.t adversaries $\mathcal{A}$ and for all $n_{\mathcal{V}} \geq 1$, $Pr[FrameGame(\mathcal{A}, \kappa, n_{\mathcal{V}}) = broken]$ is negligible (in κ) in the random oracle model.*

Proof (sketch). Assume a successful adversary $\mathcal{A}$ which breaks $FrameGame$ with non-negligible probability. From that, we construct an algorithm $\mathcal{B}$ that, given a signature oracle for an instance of the $CLS3$ signature scheme, produces an existential forgery for this instance.

$\mathcal{B}$ has to simulate the $FrameGame$ towards $\mathcal{A}$ in the random oracle model. To do so, $\mathcal{B}$ has to guess which issuer V will be "attacked" by $\mathcal{A}$. The $CLS3$ signature oracle is used by $\mathcal{B}$ for V's signatures – the keys for the other vendors and for the federation are generated honestly by the respective algorithms. If $\mathcal{A}$ corrupts a vendor different from V, $\mathcal{B}$ delivers the corresponding secret key to $\mathcal{A}$. If $\mathcal{A}$ corrupts V, the simulation fails. Assuming that $\mathcal{A}$ corrupts all vendors but one, the probability to guess the right vendor is $1/n_{\mathcal{V}}$. In [7], it is shown how to simulate the building blocks for our protocols.

In the `Issue` and `Redeem` protocols, it can be assumed that $\mathcal{B}$ can extract all secrets (by rewinding) for each PoK and SoK from $\mathcal{A}$ (it is shown in [7] that efficient knowledge extractors exist for the sub-protocols we use). Since rewinding can be done for all sub-protocols independently, $\mathcal{B}$ is still efficient.

When $\mathcal{B}$ executes `Issue` for V, $\mathcal{B}$ stores σ_i together with the signed tuple (id_i, mid, ob_i) (where id_i is obtained by knowledge extraction). This information is used to identify a forged CL signature: $\mathcal{B}$ extracts the secrets from all SoKs that are returned by $\mathcal{A}$ (in the set Π in the $FrameGame$). The condition $|CRN| > ctrC_{V,ob}$ in the $Frame$-$Game$ ensures that there are more distinct coupon IDs id_i than signatures for coupons with object ob have been queried by V. Therefore, one of the NI-SoKs π does not correspond to a coupon issued by V and $\mathcal{A}$ must have produced a forgery of a CL signature. $\mathcal{B}$ can identify the forgery using the data stored during `Issue`, and outputs it as the required existential forgery of a *CLS3* signature. Of course, this only works, if the vendor challenged by the adversary is actually the vendor V guessed by $\mathcal{B}$ at the beginning of the simulation.

Since the probability of an adversary to forge a CL signature is negligible, so is the probability of $\mathcal{A}$ to win the $FrameGame$. $\square$

To break the $ClaimGame$ (see Fig. 4), $\mathcal{A}$ successfully redeems a coupon to an uncorrupted vendor V', but V' cannot claim money for it (i.e., the `Claim` algorithm fails). In the first phase, A_1 can interact arbitrarily with the honest vendors via I_1. He must output an issuer V of a coupon (possibly corrupted) and an uncorrupted vendor V', and an arbitrary state $s_{\mathcal{A}}$ for the second phase. To win the game, A_2 must be able to redeem a coupon, allegedly issued by V, to V', but `Claim` must fail for this coupon. A_2's output $Res_{\mathcal{A}}$ is discarded.

Definition 3 (Claimability of an MCS). *An MCS is* claimable *if there is no p.p.t adversary $\mathcal{A} := (A_1, A_2)$ that can win the ClaimGame in Fig. 4 (i.e., ClaimGame$(\mathcal{A}, \kappa, n_{\mathcal{V}})$ = broken for some number of vendors $n_{\mathcal{V}} \geq 1$) with probability > 0.*

Theorem 2 (Claimability). *The proposed MCS provides claimability, i.e., for all p.p.t adversaries $\mathcal{A}$ and for all $n_{\mathcal{V}} \geq 1$, Pr[ClaimGame$(\mathcal{A}, \kappa, n_{\mathcal{V}})$ = broken$] = 0$.*

Proof (sketch). The checks in the `Claim` algorithm are a subset of the checks performed in `Redeem` by the vendor. Therefore, the condition in the $ClaimGame$ that V' accepts, but `Redeem` fails, is a contradiction (i.e., $\mathcal{A}$ can never win). $\square$

Unforgeability and unsplittability. No coalition of users should be able to redeem more coupons than have been issued by the vendors. Moreover, multi-coupons should be *unsplittable* (cf. [11]): We require that if a user U_0 shares an MC with a user U_1, as soon as one user redeems a single coupon, the other one cannot redeem any more without interacting with the user who redeemed first (note that sharing can always be achieved by copying all the data).

```
SplitGame($\mathcal{A}, \kappa, n_{\mathcal{V}}$):
    $(PK, \{SK_{V_i}\}_{1 \leq i \leq n_{\mathcal{V}}}) \leftarrow$ Setup$(1^\kappa, n_{\mathcal{V}})$;
    $(s, V, V'_{j_0}, \ldots, V'_{j_K}) \leftarrow A_1^{I'_1}(1^\kappa, PK)$
    if $K < ctrM_V$
        return unbroken;
    for $\lambda \leftarrow 0$ to $K$ do:
        $(res_A, res_v) \leftarrow$
            $(A_2(s),\ I'_1$.Redeem$_v(V, V'_{j_\lambda}))$;
        if $(res_v \neq acc)$
            return unbroken;
    return broken;
```

Fig. 5. The game defining unsplittability (*SplitGame*), where I'_1 is the interface I_1 without `Corrupt` queries.

In the games, we have to restrict the queries that are available to $\mathcal{A}$: he is not allowed to corrupt vendors, because a vendor could issue as many coupons as he likes – and hence "unforgeability with corrupted vendors" would make no sense. Moreover, we consider unsplittability to be a requirement of the entire federation. Therefore, we do not need to model corruptions: We assume that in the games defining unforgeability and unsplittability, *all users* but *no vendors* are corrupted.

Furthermore, we have to count the difference between the coupons (separately for each object ob) a vendor V issued, and the number of coupons (issued by V, with ob)

that were already redeemed, i.e., the number of coupons issued by V with object ob that are available to the adversary. Thus, a counter $ctrD_{V,ob}$ (initially 0) is introduced for each issuer V, which is increased during issue, and decreased after a successful `Redeem` (possibly at a different vendor V'). For the definition of unsplittability, it is important to know how many MCs issued by V that still contain redeemable coupons the users may have. In an unlinkable MCS, this cannot be done precisely; therefore, the MC counter $ctrM_V$ (initially 0) is just an upper bound on the users' MCs (with valid redeemable coupons). To count the MCs the users might have, $ctrM_V$ is increased by one whenever V issued a coupon. After successful redemption, the MC counter is adjusted if the number of coupons issued by V that are still available to $\mathcal{A}$ is smaller than the number of MCs (issued by the same vendor): $ctrM_V \leftarrow \min(ctrM_V, ctrD_V)$.

Interface I_1'. The modified interface I_1 without `Corrupt` queries, but with counters $ctrM_V$ and $ctrD_{V,ob}$ is denoted by I_1'.

Intuitively, to win the splittability game (see Fig. 5), $\mathcal{A}$ has to create more (in the game: $K+1$) "shares" than he has MCs (at most $ctrM_V \leq K$), which can be redeemed *independently* from each other. The state of A_2 is reset after each `Redeem` to the state s that was output by A_1; i.e., information gained in one execution of `Redeem` is not available in the other executions.

Definition 4 (Unsplittability of an MCS). *An MCS is* unsplittable *if there is no p.p.t adversary $\mathcal{A}$ that can win the SplitGame in Fig. 5 (i.e., $SplitGame(\mathcal{A}, \kappa, n_{\mathcal{V}}) =$ broken for some number of vendors $n_{\mathcal{V}} \geq 1$) with non-negligible probability (in κ).*

Theorem 3 (Unsplittability). *Assuming the security of CL signatures against existential forgery, our MCS is unsplittable, i.e., for all p.p.t adversaries $\mathcal{A}$ and for all $n_{\mathcal{V}} \geq 1$, the probability $Pr[SplitGame(\mathcal{A}, \kappa, n_{\mathcal{V}}) = broken]$ is negligible (in κ) in the random oracle model.*

Proof (idea). We can show unsplittability by a reduction, similar to the one in the proof of Theorem 1: Assuming an adversary $\mathcal{A}$ against $SplitGame$, we construct an adversary $\mathcal{B}$ against the security of the CL signature scheme (i.e., $\mathcal{B}$ will produce an existential forgery of a CL signature). $\mathcal{B}$ has to simulate the interface I_1', and play the $SplitGame$ with $\mathcal{A}$. To do so, $\mathcal{B}$ has black-box access to signature oracles for the $CLS2$ and the $CLS3$ signature schemes (these oracles can be used in the simulation because vendors cannot be corrupted). If $\mathcal{A}$ wins the game, $\mathcal{B}$ has to come up with an existential forgery of one of the signature schemes. The simulation proceeds like in the proof of Theorem 1, and it can be proven that the counter $ctrM_V$ ensures that a forgery occurs, which can be extracted from the adversary by rewinding. In this way, $\mathcal{B}$ produces an existential forgery of one of the CL signature schemes. □

In the unforgeability game, the adversary $\mathcal{A}$ can interact with the system via I_1', and he has to output the identity of an arbitrary vendor, an object ob of his choice. If more coupons (with object ob) issued by this vendor have been redeemed than the vendor originally issued (i.e., $ctrD_{V,ob} < 0$), $\mathcal{A}$ wins. Due to space restrictions, we omit the formal definition, theorem, and proof (which are analogous to unsplittability).

Unlinkability. To ensure privacy and anonymity of the customers, we require that the vendors should not be able to link a `Redeem` procedure of a customer to the corresponding `Issue` procedure, nor to another `Redeem` procedure where the customer used the same MC. Unlinkability for one user has to be provided against a collusion of vendors and other users.

Informally, unlinkability is achieved because the vendor's knowledge about elements of a single coupon depends on the actual procedure. During `Issue`, id and fid_0 are hidden, whereas mid, σ, σ'_0 are known to the vendor. During `Redeem`, id and fid_0 are disclosed to the vendor, but mid, σ, σ'_0 are hidden. fid_j $(0 \leq j < k)$ is hidden from the vendor during the j-th run of `Redeem`, but disclosed during the $(j+1)$-th run; σ'_j $(0 \leq j < k)$ is known to the vendor during the j-th run of `Redeem`, but hidden during the $(j+1)$-th run. The objects ob are known to the vendor during both `Issue` and `Redeem`. If a certain coupon object is unique to a user, this could be used for linking. Hence, the formal definition has to exclude "trivial linking" by objects. But if there are more users with coupons of a given object, it cannot be used for linking. We assume that in a system with many users, for each object there should be several users with a corresponding coupon. Hence, privacy should be preserved, for practical purposes.

For a formal definition, theorem, and proof (which are quite similar to [11]), we refer the reader to the extended version of this paper.

4 Related Work

Syverson et al. [20] introduced the concept of unsplittability in the context of unlinkable serial transactions to discourage sharing, and suggested an extension of their scheme to implement coupon books. Later, Chen et al. [10] described the properties that a privacy-protecting MCS must provide, and proposed an unforgeable, unlinkable, and weakly unsplittable scheme. However, their construction is less practical because redemption complexity is linear in k (i.e., the number of coupons in the MC).

More recently, Nguyen [15] addressed some disadvantages of [10], and defined a security model for MCSs, followed by an efficient construction based on a verifiable pseudorandom function and bilinear groups. Its issue and redeem complexity is constant w.r.t. k, it offers the same security properties as in [10], and adds a new feature to *revoke* MCs. One drawback the schemes from [15, 10] is that every issued MC must contain the same number of coupons, i.e., k is a system parameter fixed for all MCs. This limitation, as pointed out in [15], can be overcome in both schemes at the cost of efficiency, by extending the issue protocol in a way that MCs with fewer than k coupons can be issued. Another drawback of these schemes is that they do not provide coupon objects (or coupon types [8]), and they support only one vendor.

Finally, a privacy-protecting MCS scheme with strong protection against splitting has been proposed in [11]. In this scheme, the number k of coupons in an MC can vary with different MCs. Moreover, coupon objects are supported, and the proofs for the security (unforgeability, unlinkability, and unsplittability) are sketched. However, all coupons in an MC must be redeemed in a sequential order that has to be fixed during the issue protocol, and only a single vendor is considered.

As explained in [10, 15], most related schemes (e.g., e-cash, digital credentials) cannot be employed as privacy-protecting unsplittable MCSs because they have different usage patterns [18, 4], are inefficient in this setup [16], or lack at least one of the required properties [5], in particular unsplittability. Some e-cash systems (e.g., [6]) can be used as unlinkable or at least anonymous MCSs (cf. [8]). However, they are at most weakly unsplittable. Although [6] provides logarithmic issue complexity (and size) in k, it cannot support individual attributes per coupon. If coupon objects would be introduced, the issue complexity (and multi-coupon size) would also be linear in k, as in our scheme, but would not provide unsplittability.

5 Conclusion and Future Work

In this paper, we proposed a generic security model for multi-coupon schemes, suitable for a federation of vendors. We designed an efficient scheme where coupons can be redeemed in arbitrary order, and which is provably secure in this model. Future work may focus on dynamic aspects of the scheme, considering the case where vendors join and leave the federation, or on the design of more efficient schemes.

References

1. Paris Visite. `http://www.ratp.info/touristes/`, as of Oct'07.
2. Roma Pass. `http://www.romapass.it/english/index.html`, as of Oct'07.
3. Vienna Card. `http://www.wienkarte.at/EN/?l=e`, as of Oct'07.
4. C. Blundo, S. Cimato, and A. De Bonis. Secure e-coupons. *Electronic Commerce Research*, 5(1):117–139, 2005.
5. S. Brands. A technical overview of digital credentials. research report, February 2002. Available at `http://www.xs4all.nl/#brands/`.
6. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *EUROCRYPT*, number 3494 in LNCS, pages 302–321. Springer Verlag, 2005.
7. J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *SCN*, number 2576 in LNCS, pages 268–289. Springer Verlag, 2002.
8. S. Canard, A. Gouget, and E. Hufschmitt. A handy multi-coupon system. In *Applied Cryptography and Network Security, ACNS*, pages 66–81, 2006.
9. M. Chase and A. Lysyanskaya. On signatures of knowledge. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96. Springer, 2006.
10. L. Chen, M. Enzmann, A.-R. Sadeghi, M. Schneider, and M. Steiner. A privacy-protecting coupon system. In *Financial Cryptography*, volume 3570 of *LNCS*, pages 93–108, Berlin, 2005. Springer-Verlag.
11. L. Chen, A. N. Escalante B., H. Löhr, M. Manulis, and A.-R. Sadeghi. A Privacy-Protecting Multi-Coupon Scheme with Stronger Protection against Splitting. In *Financial Cryptography 2007*, volume 4886 of *LNCS*, pages 29–44. Springer Verlag, 2007.
12. I. Damgård and E. Fujisaki. A statistically hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT 2002, Proceedings*, number 2501 in LNCS, pages 125–142. Springer Verlag, 2002.
13. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 186–194, London, UK, 1987. Springer-Verlag.
14. A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *EUROCRYPT 2004*, number 3027 in LNCS, pages 571–589. Springer Verlag, 2004.
15. L. Nguyen. Privacy-protecting coupon system revisited. In *Financial Cryptography*, number 4107 in LNCS, pages 266–280. Springer Verlag, 2006.
16. L. Nguyen and R. Safavi-Naini. Dynamic k-times anonymous authentication. In *ACNS*, number 3531 in LNCS, pages 318–333. Springer Verlag, 2005.
17. A. Odlyzko. Privacy, economics, and price discrimination on the internet. In *ICEC '03: Proceedings of the 5th international conference on Electronic commerce*, pages 355–366, New York, NY, USA, 2003. ACM Press.
18. P. Persiano and I. Visconti. An efficient and usable multi-show non-transferable anonymous credential system. In *Financial Cryptography*, number 3110 in LNCS, pages 196–211. Springer Verlag, February 2004.
19. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology, IACR*, 4(3):161–174, 1991.
20. P. Syverson, S. Stubblebine, and D. Goldschlag. Unlinkable serial transactions. In *Financial Cryptography*, number 1318 in LNCS, pages 39–56. Springer Verlag, 1997.