Design and Implementation of the *idemix* Anonymous Credential System

Jan Camenisch and Els Van Herreweghen
IBM Research, Zurich Research Laboratory
8803 Rüschlikon
Switzerland
{jca,evh}@zurich.ibm.com

ABSTRACT

Anonymous credential systems [8, 9, 12, 24] allow anonymous yet authenticated and accountable transactions between users and service providers. As such, they represent a powerful technique for protecting users' privacy when conducting Internet transactions. In this paper, we describe the design and implementation of an anonymous credential system based on the protocols developed by [6]. The system is based on new high-level primitives and interfaces allowing for easy integration into access control systems. The prototype was realized in Java. We demonstrate its use and some deployment issues with the description of an operational demonstration scenario.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption—*Public key cryptosystems*

General Terms

Design, Security

Keywords

Privacy, Anonymous Credential Systems, Cryptographic Protocols

1. INTRODUCTION

The protection of users' privacy when performing Internet or web-based transactions is an important factor in the acceptance and use of Internet and web services.

Solutions for minimizing release of personal information can be based on one of many proposed techniques for anonymizing the transport medium used between users and service providers, e.g., [26, 18, 27]. This may anonymize the user towards outsiders and, if desired, towards the service provider.

CCS'02, November 18–22, 2002, Washington, DC, USA.

Service providers may require authentication (e.g., for controlling access to resources) or accountability of users' actions, in which case users need to prove their identity, or at least possession of a certificate or capability of a certain type. Such a certificate may contain a pseudonymous identity of the user, or contain only the necessary attributes required for accessing a certain service. However, when using certificates as defined by X.509 [11] or SPKI [2], or even certificates specifically constructed for conveying policy or authorization information as in Keynote [3], different uses of the same certificate still remain linkable to each other. They can eventually identify a user through a combination of context and addressing information from one or a series of transactions. Moreover, the transaction in which the certificate was issued can be linked to the transaction where it is used and thus, it the issuer and the verifier collude, the user can identified directly.

These linkabilities can be avoided by using an anonymous credential system (also called pseudonym system) [8, 9, 12, 24]. In such a system, the organizations (service providers and credential issuers) know the users only by pseudonyms. Different pseudonyms of the same user cannot be linked. Yet, an organization can issue a credential to a pseudonym, and the corresponding user can prove possession of this credential to another organization (who knows him by a different pseudonym), without revealing anything more than the fact that the user owns such a credential.

In this paper, we describe the design and implementation of *idemix* (short for *ide*ntity *mix*), a prototype of the credential system by Camenisch and Lysyanskaya [6]. We describe the *idemix* functionality using high-level primitives. These primitives allow reasoning about security and privacy features, while hiding the complexity of the cryptographic protocols, as well as the differences between actual protocols realizing the same primitive. We also developed additional functionality for service providers and credential issuers to configure and enforce resource access control and credential issuing decisions. As we demonstrate with an example, this allows the use of the prototype in developing actual applications using concepts of anonymous and attribute-based authentication and access control.

After describing the functionality of the credential system protocols in Section 2, we describe in Section 3 the high-level primitives. Section 4 describes the architecture and implementation of the prototype implementing these protocols, as well as the additional modules developed to

support easy configuration, creation, management and deployment of *idemix*-based applications. The use of the prototype is demonstrated with an implemented scenario. Section 6 raises security and infrastructure management issues related to the deployment of *idemix*. Section 7 states conclusions and lists future work.

2. *IDEMIX* PROTOCOLS, PSEUDONYMS AND CREDENTIALS

In this section we describe the functionality of the credential system. The basic protocols for issuing and showing credentials are described in Section 2.1; Sections 2.2 and 2.3 describe optional features of protocols and credentials.

2.1 Basic Credential Protocols

The core of the *idemix* system consists of the protocols described in [6]. This section describes these protocols in terms of parametrized primitives of which functionality can be easily explained and mapped to system interfaces.

The entities in the system are users, who obtain and show credentials, and organizations issuing and verifying credentials. Another type of organization, *de-anonymizing organization*, is discussed in Section 2.3.1[1]. Thus, a user U can obtain a credential C from an (issuing) organization O_I; and then show the credential C to another (verifying) organization O_V. A credential is always issued on a pseudonym N under which U is registered with (or known by) the issuing organization O_I. A credential may have certain attributes (`attr`). When showing a credential, the user can choose which of the credential's attributes shall be revealed (see Section 3.4).

Pseudonym registration, credential issuing and credential verification are interactive protocols between the user and the specific organization. A user U has a (single) master secret S_U, which is linked to all the pseudonyms and credentials issued to that user. Issuing and verifying organizations all have a public/private key pair. The organization issuing a credential uses its private key to generate the credential; the credential can then be verified using the issuing organization's public key, either by the user when receiving the credential, or later on by any organization to which the user shows the credential. When showing a credential, the user uses the public key of the verifying organization which, in turn, needs its private key in the protocol.

Obtaining a credential from O_I and showing it to O_V works as follows (cf. Figure 1). First, U contacts O_I and establishes a pseudonym N with O_I. If N is eligible to get a credential with an attribute *attr*, O_I produces a credential C by signing a statement containing *attr* and N and sends C to U. Now U can show this credential to O_V. That is, using a zero-knowledge proof, U convinces O_V of (1) possessing a signature generated by O_I on a statement containing *attr* and N, and (2) knowing the master secret key S_U related to N. We stress that U does not reveal any other information to O_V. In particular, U does not send O_V the actual credential. This way of showing a credential together with the zero-knowledge property of the proof ensures the unlinkability of different showings of a credential and also the unlinkability of a showing of a credential to the pseudonym to which the credential was issued. This means that U can show C to O_V (or any other verifier) an unlimited number of times, without these credential shows becoming linkable to each other or to a pseudonym. (Exceptions are one-show credentials, which are discussed in Section 2.2). This unlinkability is maintained even if O_V and O_I are the same organization (or pool their data).

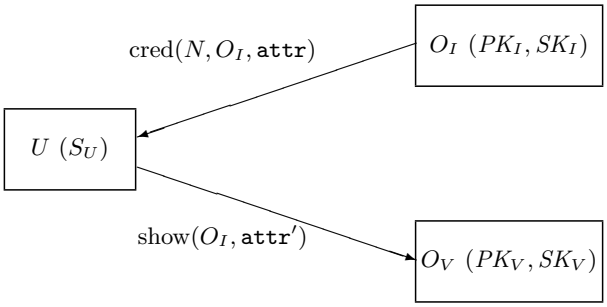

Figure 1: Basic credential system protocols.

Note that from this unlinkability property it follows that the user is anonymous towards the verifying organization. Of course, this property of the pseudonym system can only provide real anonymity to the user if the communication channel used supports anonymity [7, 18, 26, 27].

While, in general, this approach to showing a credential is not very efficient, the special signature scheme used by the credential system [6] allows for an efficient realization of the zero-knowledge proof described above. In fact, as indicated by our performance results, the computational complexity for both the user and the verifying organization executing the protocol for showing a credential corresponds to generating a small number of signatures in the standard RSA signature scheme.

As all of a user's credentials are linked to his master secret, sharing a credential would imply also giving away one's master secret. This not only ensures that users cannot pool their credentials (e.g., to obtain a new credential) but also allows the implementation of measures to discourage users from sharing their credentials. One way to do this is *PKI-assured non-transferability*, where the user's master secret key is tied to some valuable secret key from *outside* the system (e.g., the secret key that gives access to the user's bank account) [13, 17, 24]. Thus sharing a credential implies also sharing this valuable secret key. However, such a valuable key does not always exist. An other, novel way of achieving this is *all-or-nothing* non-transferability [6]. Here, sharing just one pseudonym or credential implies sharing all of the user's other credentials and pseudonyms in the system, i.e., sharing *all* of the user's secret keys *inside* the system.

In cases where the verifier and the issuer are the same entity, sharing credentials can be limited by the approach proposed by Stubblebine, Syverson, and Goldschlag [28]. In this approach a credential can only be used once, but each time a credential is used, a new credential is issued. Thus, when a credential is given away, only the person using the credential first is given the next credential. This mechanism makes sharing access to a resource tedious.

Using the so-called Fiat-Shamir heuristic [16], the protocol for showing a credential can also be turned into a sig-

[1]In the remainder of the text, *organization* is used for credential issuing and/or verifying organizations. Unless explicitly mentioned, it does not include *de-anonymizing organizations*.

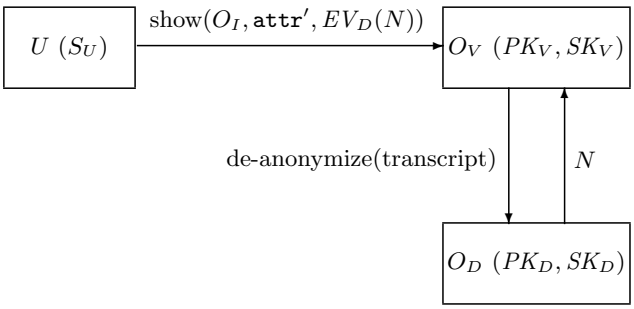

Figure 2: De-Anonymization

nature scheme. The meaning of a signature will then be "a person possessing a credential issued by O_I has signed this message."

Both *all-or-nothing* non-transferability as well as the signature functionality will only be implemented in a future version of the prototype.

2.2 Credential Options and Attributes

Credentials can have options (such as one-show, or multi-show) and attributes. The one-show credentials incorporate an *off-line* double-spending test [10]: when showing a one-show credential more than once (to the same or different organizations), this results in transcripts from which the issuing organization can extract the pseudonym N on which it was issued.

Examples of credential attributes can be an expiration date, the user's age, a credential subtype. When showing a credential, the user can choose which attribute(s) to prove something about, and what to prove about them. E.g., when showing a credential that has attributes (`exp-date = "2002-05-19", age = 55`), the user can decide to prove only that `age > 18` (see also Section 3.4).

2.3 Parameters of the Show Protocol

2.3.1 De-Anonymizible Show of a Credential

De-anonymization mechanisms allow to reveal the identity of a user (*global de-anonymization*, also called *global anonymity revocation*) or to reveal a user's pseudonym with an issuing organization (*lobal de-anonymization* or *local anonymity revocation*). *Global de-anonymization* allows for global accountability of transactions (e.g., for identifying a user performing illegal transactions); *local anonymity revocation* can be applied by the issuing organization to take measures when a user misuses his credential.

Both types of de-anonymization are optional and require U's cooperation when showing a credential. They require the existence of a designated third party, a *de-anonymizing organization* O_D (see Figure 2). O_D has a public-private encryption-decryption key pair (PK_D, SK_D). Using this variant of the show protocol, U encrypts N with O_D's public encryption key. This encryption is verifiable (denoted $EV_D(N)$), which means that O_V has proof that O_D can decrypt and reveal the relevant N from O_V's show protocol transcript. There may be several de-anonymizing organizations in the system, from which U may be able to chose. By including also a de-anonymization condition, U can decide under which condition he consents to the transcript being de-anonymized. Later, when deemed necessary by O_V, O_V can send the transcript to O_D. O_D can then decide whether this condition is fulfilled and, if so, de-anonymize the transcript and returns N (*local de-anonymization*).

Global de-anonymization uses essentially the same technique. It requires, in addition, the existence of a special credential issuing organization, a *Root Pseudonym Authority*, which only issues credentials on pseudonyms of which it knows the mapping with a real user identity. A user typically has a single pseudonym (root pseudonym) with the Root Pseudonym Authority, and one credential (root credential) on that root pseudonym (additional pseudonyms or credentials with the Root Pseudonym Authority would anyway be linkable to the user).

2.3.2 Showing a Credential Relative to a Pseudonym

Using this option, U, who has obtained a credential C by O_I on N_I, and who is known under pseudonym N_V to O_V, proves possession of C to O_V, while also proving that the pseudonym to which C was issued belongs to the same user as does N_V. More precisely, the user proves that the same master secret key S_U that is linked to N_V is also linked to the credential C and the pseudonym (N_I) the credential C is issued on.

This option is mandatory for U to convince O_V of possession of several credentials. Without using the option, two users each possessing a different credential could each show their credential to O_V and fool O_V into believing that it talked to a single user possessing both credentials.

Furthermore, this option is also mandatory if showing of a credential is a precondition for a user to get another credential. The reason for this can be seen from the following example. Let us assume that U wants to obtain a credential from O_{VI}; O_{VI}, in order to issue such a credential, requires U to show a credential by O_I. If U has such a credential, he first registers a pseudonym N_{VI} with O_{VI}, and then shows the credential by O_I to O_{VI}, upon which O_{VI} considers the precondition to be satisfied and issues the new credential on N_{VI}. If U has no such credential, he can try to collaborate with U' (who does own the credential) by asking U' to perform the second step (showing the credential by O_{VI}). And indeed, if O_{VI} does not require U to show the O_I credential relative to a specific pseudonym, U will obtain the credential from O_{VI} without fulfilling the precondition. By requiring to show the O_I credential relative to N_{VI}, O_{VI} enforces that the same user who showed the O_I credential gets the new O_{VI} credential.

3. CREDENTIAL SYSTEM PRIMITIVES

In this section, we start out by describing representations for pseudonyms and credentials, and then define representations of credential attributes and protocol options. Subsequently, we describe the high-level primitives of the pseudonym system.

3.1 Pseudonyms

A pseudonym N of user U with O_I cannot be mapped to a data representation shared by U and O_I: U has a secret value (other than the user's master secret) attached to each pseudonym N - knowledge of this secret value is required to make any operation with pseudonym N, such as obtaining a credential. Thus, an implementation of the credential system needs different representations (or data types) for

a pseudonym, depending on the role of the actor (user or organization).

Nym(N, Ns, U, O, X) could be an abstract representation of a pseudonym of user U with organization O. N is common to U and O and uniquely identifies the pseudonym to both U and O. Ns is the secret value associated by the user with the pseudonym. X is a statement or set of statements which O attaches to N: information obtained during registration (e.g., a real user identity in the case of root pseudonym registration) as well as up-to-date information about credentials issued to N.[2]

Nym(N, Ns, U, O, X) is represented in reality by two sets of data:
UserNym(N, Ns, O), and OrgNym(N, X).[3]

Note that the user's master secret S_U, though essential in using the UserNym, is common to all a user's pseudonyms and is not considered part of the UserNym representation. Note also that the statement X is application-specific, and is not explicitly supported by the core of the pseudonym system as it is introduced in Section 4.

O does not authenticate to U during pseudonym registration. While this is not a security threat, as the registration does not reveal any information about U, S_U or Ns, the registration of a pseudonym with an impersonator of O can lead to denial of service when U then tries to obtain a credential on N from O. For this and other reasons discussed in Section 6, we assume every O to have a certificate with which to authenticate communication with users.

3.2 Credentials

Credentials have a different representation at the user and at the organization side as well: when a credential C is issued by O_I on N, user and organization get to store different values associated with it.

Cred(N, Ns, U, O_I, C, T) represents a credential, issued by organization O_I on pseudonym N of user U. T represents the credential's type (CredInfo), including the specific issuing key (PK_I) options and attributes:

$$\text{CredInfo}(PK_I, \text{MultiShow}, \text{Expiration}, \text{Subtype}, \text{Age}, ...).$$

Using a similar reasoning as for pseudonyms, Cred(N, Ns, U, O_I, C, T) expresses a relationship between UserCred(UserNym, C, T, O_I) and OrgCred(OrgNym, T).

3.3 CredShowFeatures

The parameters of a credential showing are expressed in a CredShowFeatures parameter array. The parameter RelNym indicates whether the show is relative to a pseudonym known to the verifying organization (in which case an additional argument N_V will indicate this pseudonym). Another parameter contains de-anonymization information specifying whether, under which condition and by which de-anonymization organization O_D (or which public key PK_D) the show transcript will be de-anonymizable (*local* or *global* de-anonymization):

$$\text{CredShowFeatures}(\text{RelNym}, [PK_D, \text{ConditionLocal}], [PK_D, \text{ConditionGlobal}])$$

[2]Nym(N, Ns, U, O, X) is the equivalent of following statement: The value N is known by U and O to be a valid pseudonym according to the credential system specification. This means that there has been a pseudonym registration procedure between U and O, during which both U and O contributed to the value of the pseudonym, and during which O verified that N is based on a well-formed secret S_U. During that registration procedure, O associated X with N. U's contribution is linked to Us master secret S_U as well as to the new pseudonym-specific user secret Ns in a way that credentials issued on N are linked to these secrets.

[3]The term DataType(Field1, Field2, ...) informally defines the contents of a data set of type DataType.

Primitive	Inputs/Outputs
U_registerNym	IN: UserSecret OUT: UserNym
O_registerNym	IN: - OUT: Transcript (incl. OrgNym)
U_getCredential	IN: UserSecret,UserNym, CredInfo OUT: UserCred
O_issueCredential	IN: OrgKeys, OrgNym, CredInfo OUT: Transcript (incl. OrgCred)
U_showCredential	IN: UserSecret, UserCred, CredInfo, CredShowFeatures, [UserNym] OUT: -
O_verifyCredential	IN: OrgKeys, CredInfo, CredShowFeatures, [OrgNym] OUT: Transcript
O_checkDoubleSpending	IN: OrgKeys, Transcript, Transcript OUT: OrgNym
DO_deAnonymize	IN: DeAnOrgKeys, Transcript OUT: OrgNym

Figure 3: Protocol Primitives

3.4 Protocol Primitives

Figure 3 lists the basic protocol primitives for registering pseudonyms, issuing and verifying credentials, verifying double-spending of one-show credentials, and de-anonymizing. Primitives invoked by a user (respectively org, de-anonymizing org) carry the prefix U_ (O_, DO_).

In the user-invoked (U_) primitives, the identifier of the targeted organization (O_I, O_V) is not listed as parameter: it is assumed that the application calling the user primitives has set up a communication channel with the correct organization, using addressing information obtained at application level.

The organization (O_) versions of these primitives result in a Transcript of the protocol which, among the cryptographic transcript and other application data, includes also the newly established OrgNym or OrgCred. The calling application is responsible for extracting this information and store it in appropriate persistent storage.

When showing a credential with CredInfo T, a user can choose which subset of attributes he wants to prove, and what to prove about them, by setting the CredInfo parameter T' in the UserShowCredential() primitive. This T' is communicated to the verifying organization and used as CredInfo parameter also in OrgVerifyCredential(). E.g., if

$$T.\text{Age}=\text{'55'},\ T.\text{Expiration}=\text{'20020831'},$$
$$T'.\text{Age}=\text{'>18'},\ T'.\text{Expiration}=\text{'any'}$$

then the show protocol will only prove that T.Age>18.

Before executing the show protocol, the calling user application should verify whether canFulfill(T, T') holds.

The de-anonymization primitive does not check the condition for de-anonymization (which is included in the transcript to be de-anonymized): this is assumed to be the application's responsibility.

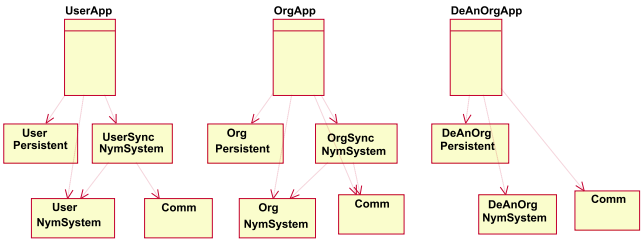

Figure 4: User, Org, and DeAnOrg components

Additional primitives, such as for generating parties' key pairs and master secrets (and extracting the public information from them), are described in the next section.

4. THE *IDEMIX* PROTOTYPE

In this Section, we describe design and implementation (Java) of the *idemix* prototype. The core of the *idemix* system is the NymSystem package as described in [5] implementing the UserNymSystem, OrgNymSystem[4] and DeAnOrgNymSystem components in Figure 4. Each of these components offers functionality related to the specific cryptographic operations executed by the different entities, as well as methods to create a new instance of the entity by generating cryptographic key material (user master secret, organization's public/private key pair, de-anonymizing organization's public/private encryption key pair). The following paragraphs shortly discuss the different interfaces.

4.1 OrgNymSystem and UserNymSystem Token-Based Interfaces

UserNymSystem and OrgNymSystem contain the user's and organization's methods to compose and analyze the cryptographic tokens exchanged in the nym registration, credential issuing and credential show protocols. OrgNymSystem, in addition, contains a method for verifying whether two show transcripts result from the double-spending of a one-show credential, and if so, extract the pseudonym the credential was issued on. It is the calling application's responsibility to call this method when deemed appropriate.

Each of the interactive user-organization protocols is implemented by a user (in UserNymSystem) and an organization (in OrgNymSystem) state machine (encapsulated in a UserProtocol or OrgProtocol) which the calling application initializes (e.g., initRegProtocol). After this, the calling application can execute the protocol using the state machine's getNextMsg() method and a communication channel. When the protocols is finished, the state machines (UserProtocol or OrgProtocol) allow the calling application to retrieve a newly formed UserNym or UserCred (OrgNym or OrgCred). The advantage of this token-based interface is that the protocols can be used asynchronously.

For easier programming of synchronous applications, synchronous interfaces (UserSyncNymSystem and OrgSyncNymSystem) (see Section 4.2) were implemented hiding the protocol state machines and the transport of cryptographic tokens from the calling application.

[4]NymSystem does not distinguish between verifying and issuing organizations. A distinction can be made only by not enabling an organization to issue credentials using the RequestGranter (Section 4.6.2) functionality.

The NymSystem library is stateless and consists of static methods: i.e., the calling application is responsible for providing all the cryptographic information (including system parameters, such as key lengths, and the actual key material) when calling the library methods[5]. E.g., when a user master secret is generated using createUser(), the result (UserNymSysData) contains the master secret as well as the system parameters, and is a parameter to any of the cryptographic protocols. For ease of application programming and to avoid repeated file access for key material by UserApp and OrgApp, the synchronous extensions which are described in the next section were given a non-persistent state: the classes implementing them contain static variables with handles to system parameters and key material. The organization's public key can be extracted from the OrgNymSysData (and distributed to users upon system setup or update).

4.2 UserSyncNymSystem and OrgSyncNymSystem Synchronous Interfaces

The token-based interfaces of UserNymSystem and OrgNymSystem leave the task of driving the protocols to the calling application. It is also the calling application's responsibility to communicate the meta-information that allows user and organizations to initialize their respective protocol state machines; e.g., it is up to the calling user application to communicate to the organization which kind of credentials and with which options/features it wants to prove possession of, or what type of credential or data it wants to get.

To enable easy programming of synchronous applications on top of this token-based interface, the synchronous interfaces (Figure 5) UserSyncNymSystem and OrgSyncNymSystem take care of the signalling of meta-information as well as of driving the protocol state machines. They require the respective UserApp and OrgApp to pass a communication (ClientCommSession, ServerCommSession) object. This allows the calling application to decide whether to create a new communication session for the exchange, or to re-use a communication channel in use by the application; it also allows the calling application to decide which protocols are run within the same communication session. This allows maximal flexibility when hooking *idemix* as an authentication mechanism into an existing application.

4.3 DeAnonOrgNymSystem

A deanonymizing organization does not carry out interactive protocol. It may receive a transcript to be deanonymized from any out-of-band communication channel, and may be operating in batch or asynchronous mode. Therefore, DeAnOrgNymSystem only has an asynchronous interface (Figure 5): it provides methods for creation of the organization and for de-anonymizing a transcript. Also here, the public key in DeAnOnOrgNymSysData (resulting from the createDeAnOrg()) can be extracted in order to be distributed to users.

4.4 Communication

ClientCommSession and ServerCommSession are defined as interfaces offering generic read and write methods. It is up to the calling applications to pass a communication

[5]This differs slightly from the earlier version described in [5], where NymSystem was parametrized with file locations of secret keys

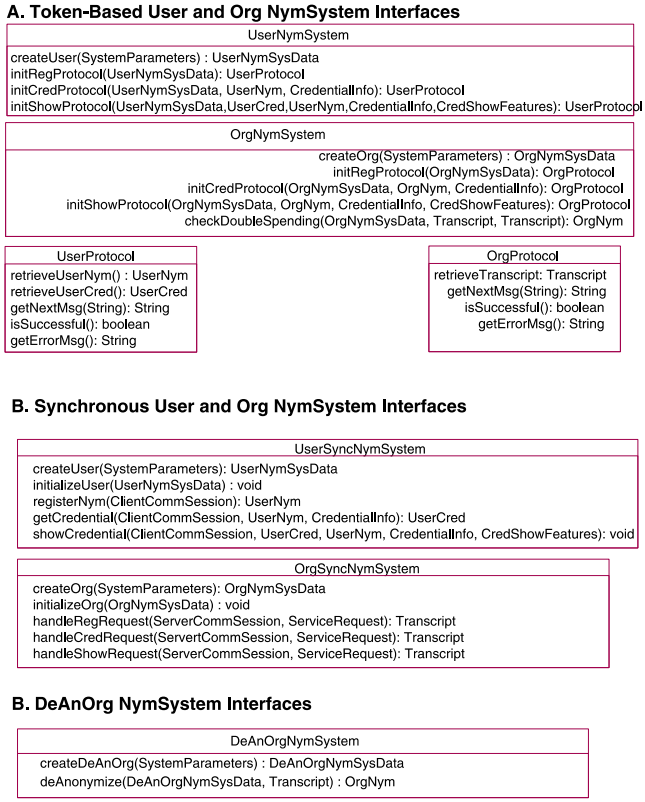

Figure 5: User, Org and DeAnOrg Token-Based and Synchronous interfaces

object that implements these interfaces. Use of encrypted or authenticated channels (e.g., using SSL) is allowed but not mandatory; the security of the communication channel is discussed in more detail in Section 6.

Our prototype SSLClientCommSession and SSLServerCommSession implementation of these interfaces use a proprietary Java SSL implementation. Organizations' address and SSL information is part of the public information created by an organization's initialization program, distributed to the users and stored in the users' persistent storage (see Section 4.5). Communication is authenticated (organization to user) and encrypted, and the user can check the certificate using the getPeerCertificate() method of the SSLClientCommSession.

4.5 Persistent Data Storage

Credentials, pseudonyms, master secrets, organizations' keys, system parameters, and address information have to be stored persistently to have a workable prototype. We defined interfaces allowing the various entities to store and retrieve this information. These interfaces, the search keys used for retrieving information about credentials, organizations, pseudonyms etc., and how the persistent information is organized (one or several databases, password-protection, etc.) ultimately depend on how the application will use the system, and are not the focus of this work.

Each of the interfaces used in our prototype (UserPersistent, OrgPersistent, DeAnOrgPersistent) combine access methods to the appropriate data sets for each entity. The example scenario in Section 5 illustrates how our example applications use the persistent data.

4.6 Building Applications: Granting and Processing Requests

When building real applications, we have to link application with the NymSystem. As granting requests for both credentials and data can depend on a user having shown one or several credentials, granting and processing of credentials and other resource requests are both treated here as actions dependent on access control conditions. In this section we give an overview of how we defined the rules governing the organizations' access control conditions, and how we defined the modules granting and processing service requests. This was done in such a way that they can be tailored to a specific application, and that processing conditions and resources not known to the *idemix* system can be added and linked to the *idemix*-specific ones.

In the following discussion, we refer to Figure 6, where OrgApp stands for an organization application receiving and processing requests. MyOrgPersistent allows OrgApp to store and retrieve its key material OrgNymSysData (input parameter to OrgSyncNymSys methods), to the organization's Rules (input parameter to the RequestGranter), to the organization's Transcripts file (where the credential protocol transcripts, resulting from the OrgRequestProcessor, are stored persistently). An OrgApp may have one or more OrgSession threads which accumulate the Transcript information on a specific communication session with a user.

4.6.1 Organizations' Access Rules

An organization has to specify which condition or conditions a user has to fulfill in order to get access to data or to get a credential. A Condition can require to show a credential:

ShowCondition(CredInfo, CredShowFeatures)

expresses what type of credential U needs to show, and using which options or parameters. It can be used by UserApp and OrgApp to parameterize a NymSystem credential show.

Alternatively, a Condition can be an *idemix*-external fact, expressed in an ExternalCondition:

Condition(ShowCondition | ExternalCondition)

The format of an ExternalCondition is defined by the application, and its fulfillment is verified using an application-provided method (see RequestGranter and extensions, Section 4.6.2). E.g., an OrgApp may grant an anonymous newspaper subscription credential based on a (non-anonymous) proof of credit card payment. The checking of this proof is a condition out of the scope of *idemix* and has to be implemented by the application programmer in an extension of the RequestGranter class.

An organization' Rules set is a collection of Rule entries ([] is used as an array notation):

Rules(Rule[])

Each Rule consists of the description ResourceDescription of the resource(s) for which this rule is valid, and a set of conditions to be fulfilled for accessing the resource:

Rule(ResourceDescription, Condition[])

A ResourceDescription can describe a credential or an external resource; an external resource (ExternalResource) is

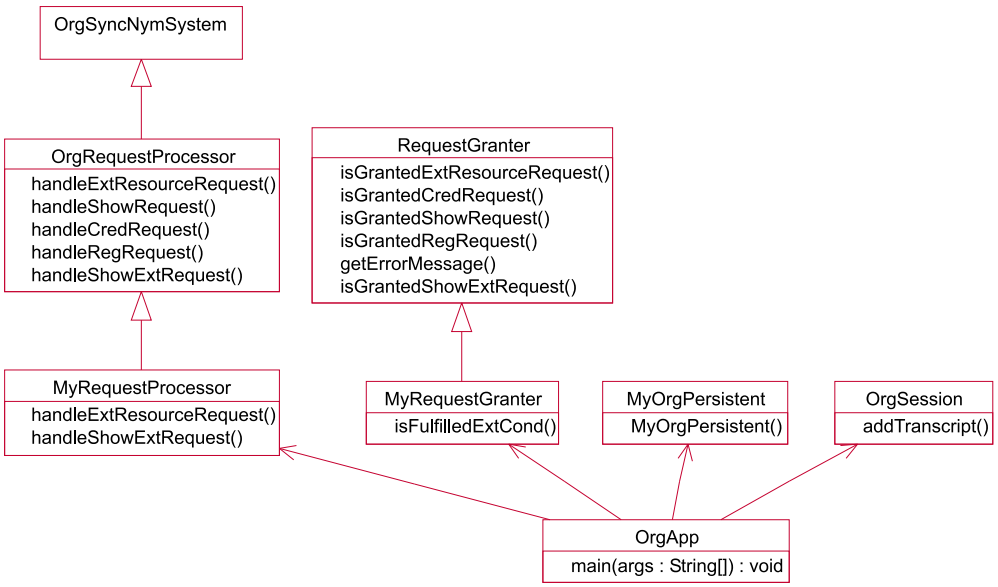

Figure 6: An Organization Application

any resource other than a credential (e.g., a URL). It has an application-defined format.

ResourceDescription(CredentialInfo | ExternalResource)

Granting a pseudonym or verifying a proof are currently defined to be unprotected resources; the ResourceDescription could easily be extended to make also these requests subject to conditions.

In an operational system, rules and conditions can be communicated dynamically (as a result of a resource request) from OrgApp to UserApp; or it can be part of the organization's publicly distributed information and stored by the users (as described in Section 4.5). They parameterize the granting decision of the organization's RequestGranter (Section 4.6.2).

4.6.2 *RequestGranter*

An extendable RequestGranter class contains default implementations of the methods that grant different requests. Calls to the RequestGranter class are parametrized with the Rules set, the persistent Transcript storage (Which credentials did the user owning this nym show in the past?) and with non-persistent OrgSession storage (Which credentials did the user show in this session?)

The default RequestGranter cannot evaluate ExternalConditions and will consider them unfulfilled by default. An application can provide additional methods in its own extension of RequestGranter (MyRequestGranter in Figure 6), overriding the isFulfilledExtCond() method of RequestGranter.

4.6.3 *OrgRequestProcessor*

A default OrgRequestProcessor module, extending OrgSyncNymSys, deals with incoming requests and extracts the appropriate arguments for the OrgSyncNymSys methods. This default OrgRequestProcessor does not know how to handle application-specific resource requests or requests that deal with fulfilling an ExternalCondition (e.g., showing a credit card receipt). An application-specific extension (MyOrgRequestProcessor) can override the (by default failing) handleExtResourceRequest() and handleShowExtRequest() methods.

4.7 Performance

Protocol	Options	Time (sec)
RegNym		0.2-0.3
GetCred	any option	3.4-4.9
ShowCred	no option, or w.r.t. pseudonym	7.8-8.2
ShowCred	+ one show (on-line or off-line)	+ 0.6-1.0
ShowCred	+ exp date	+ 2.9-3.2
ShowCred	+ local revocation enable	+ 6.5-7.2
ShowCred	+ local revocation enable	+ 6.5-7.2
ShowCred	all options on	24.8-25.3

Table 1: Performance using 1024 bit moduli.

Table 1 lists the execution times of the different operations. The measurements were made on IBM T23 laptop machines (1.1 MHz Pentium III) running Debian Linux using Java 1.3.1 (Blackdown). The "+"-signs in the ShowCred entries mean that if one switches on an option in the ShowCred protocol, then execution time will increase by the given time.

These execution times are for a preliminary version of the NymSystem where no optimization for multi-based exponentiation is used. We are currently implementing such optimizations. First tests indicate that a speed-up by a factor of about 4-5 can be obtained. Furthermore, the cryptographic protocols are currently such that first the users does lots of computations, sends the result to the verifying organization, and then the verifying organization does lots of computations. We plan to optimize the protocols in this respect also, which should provide a speed-up by a factor of a little less than 2.

5. AN EXAMPLE SCENARIO: AN ANONYMOUS SUBSCRIPTION TO THE NEW YORK TIMES

In this section, we demonstrate the use of the prototype by user and organization applications. We define four organizations: a Root Pseudonym Authority (PA), a bank (ARGENTIX), the New York Times news subscription service (KIOSK), and the New York Times news service (NYT). NYT serves items in its cartoons section only upon verification of a subscription credential issued by KIOSK; KIOSK, in turn, issues such a credential upon verification of a (one-show) $10 credential; ARGENTIX issues such a credential based on proof of an (non-anonymous, *idemix*-external) payment, combined with the verification of a PA root credential. PA unconditionally grants root pseudonyms and credentials (In a more realistic scenario, a user could be required to show an external certificate when registering a root credential, as discussed in Section 6.3).

5.1 Creating and Configuring the User and Organizations

A demo setup program uses the NymSystem user and organization creation facilities to create one user and four organizations. It assigns IP addresses and port numbers to the four organizations, as well as SSL Certificates which are created using the KeyMan [14] PKI management tool. It also creates rules for the three organizations (see below). The initialization program creates persistent data sets for each of the four entities, and initializes each of the organization's data sets with its own OrgNymSystemData key material. The user's data set is initialized with the user's UserNymSysData key information, as well as all the organizations' public information (*idemix* public key, addresses, SSL certificates, rules).

PA and KIOSK use the default RequestGranter and OrgRequestProcessor as they do not deal with ExternalConditions or ExternalResources; ARGENTIX implements its own ArgentixRequestGranter defining the verification of the credit card receipt; NYT, finally, implements its own NYTRequestProcessor with handleResourceRequest() mapping a resource request (URL) into the actual contents of a web page.

5.2 User Credential Manager and Browser Plug-In

Based on the *idemix* prototype, [25] describes the design and implementation of a Credential Manager implemented as a plug-in to a WBI [1] browser proxy. Figure 7 shows an instance of the Credential Manager in a scenario with the four organizations initialized as described above. This Credential Manager popped up after a user entered a "http://-www.nyt.com/cartoons" URL in his browser URL window. The Credential Manager then allows the user to view the relevant condition tree applying to the request, the conditions for which he has the necessary credential or external proof (tick-off symbol) and the credentials he already owns in his credentials purse. E.g., he already has a credential from PA.

The two conditions by ARGENTIX are related to (1) a ShowCondition: showing the credential from the PA, and (2) an ExternalCondition giving a reference to a credit-card payment. This reference is implemented by, e.g., a serial number of the payment. The ExternalCondition shows up in the condition tree; but as the payment reference is not an *idemix* credential, there is no corresponding credential in the credentials purse.

When clicking on a condition in the tree, the details are shown in the selected condition window, e.g., the KIOSK requires a one-show credential (multi-show = false) issued by ARGENTX with subtype = 10. It also allows the user to chose local identifiers (e.g., "kiosknym") for the pseudonyms he establishes with the different organizations, and to GET and SHOW credentials. After fulfilling all the conditions, the requested contents (cartoons page) show up in the browser window.

6. DEPLOYMENT CONSIDERATIONS

In this section, we discuss some issues related to the deployment of *idemix*.

6.1 Deploying *idemix* as a Privacy-Enhanced Public-Key Infrastructure with External Certification

In an operational system, public information about organizations (whether or not regularly updated) needs to be certified: users need authenticated information about where to get or show a credential, what is the *idemix* public key of an organization, and what is its SSL certificate. Also, a real Root Pseudonym Authority can only guarantee total accountability (global anonymity revocation) if a user's real-world information was authenticated upon registering the root pseudonym.

A deployment environment using *idemix* credentials as a (privacy-enhanced) Public-Key Infrastructure needs to provide hooks for an external Public-Key infrastructure (PKI). In this external PKI, users and organizations have public-key certificates issued by a Certification Authority. We call this authority Certifix, although it may be an existing Certification Authority; the only requirement being that it can issue organizations' "*idemix* certificates" certifying the whole set of an *idemix* organization's authenticated information. Depending on implementation and deployment choices, such an organization's *idemix* certificate may contain *idemix* keys, address and SSL information, and access rules.

Users also have Certifix certificates and use them to authenticate "real-world" information during root pseudonym registration.

6.2 The Role of Authenticated Communication in Linking Transactions Based on *idemix* Authentication

Authenticated communication (e.g., using SSL server authentication) allows users to authenticate organizations with which they register a pseudonym, to which they show a credential or from which they obtain a credential. When several protocol executions (including application-level resource requests) are linked by an authenticated communication channel, this also allows servers to securely link *idemix* authentication (who showed the correct credential) with providing the resource (who gets the data).

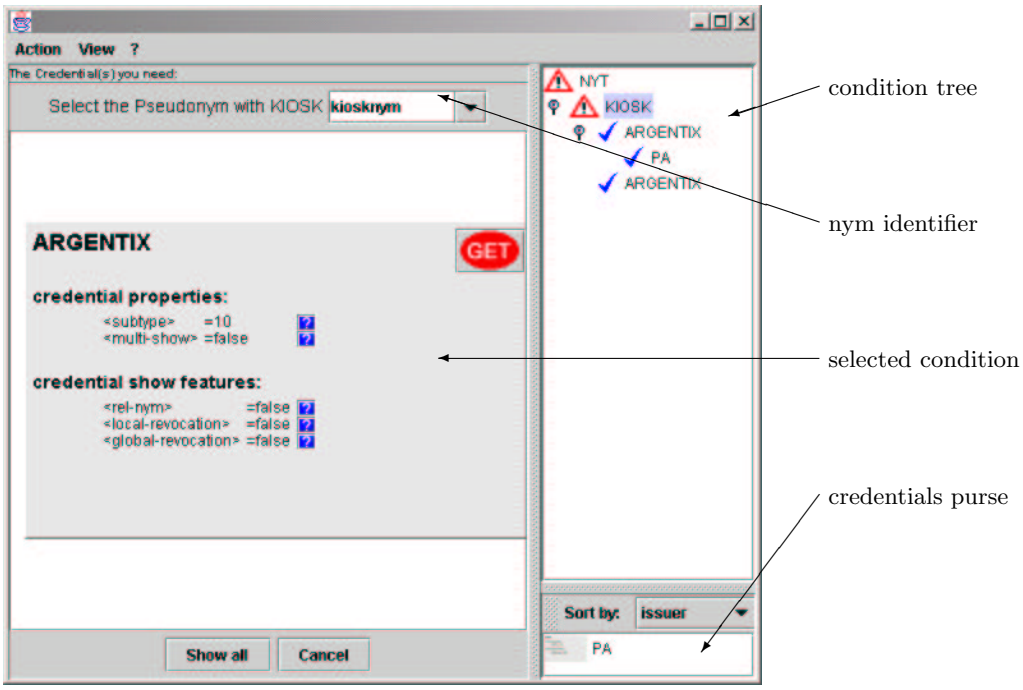

Figure 7: UserCredential Manager

6.3 Infrastructural Issues: User Registration and Organization Updates

In a real deployment environment, users and organizations dynamically join the system, and organizations may periodically update public information such as rules, public keys, addresses, or SSL information (their *idemix* certificates).

A user joining the system may not only need to authenticate using their real-world certificate when registering a root pseudonym with the root pseudonym authority; he may also have to prove registration (or payment of a license). This may be realized by the Root Pseudonym Authority checking an additional condition.

Also, organizations' *idemix* certificates need to be distributed and updated in an efficient way. A separate InfoServer entity may serve as a central repository for up-to-date organizations' *idemix* certificates. Organizations post their *idemix* certificates to the InfoServer; a certificate update may update whole or part (e.g., only new rules set) of an organization's *idemix* information. Revocation issues may be dealt with by Certificate Revocation Lists (CRLs) issued by the InfoServer; or avoided by issuing short-lived *idemix* certificates.

6.4 *Idemix*, Trust Management and Attribute-Based Access Control

Decentralized trust management, a term introduced by Blaze, Feigenbaum and Lacy [4], deals with access control and authorization in distributed environments. Different trust management systems and languages have been proposed, e.g., [3, 21, 20, 19, 23, 22, 15]; a credential or certificate modeled by those systems binds a public key to attributes and/or authorizations. Access control and trust establishment policies controlled by resource owners allow authorization decisions based on these attributes and authorizations, or on derived role assignments. Trust between the verifier and the issuer of a credential can be modeled through delegation of attribute authority, which allows a resource owner to delegate authority over an attribute to another entity. Some work also deals with automatic collection or discovery of (part of) certificate chains (e.g., [23, 22, 19]).

The access control rules and conditions language introduced in Section 4.6.1 was designed to illustrate the capabilities and usage of *idemix* for configuring anonymous attribute-based access control in a prototype application environment. However, as *idemix* certificates can be used to formulate any assertion (also identity assertions, if required), *idemix* attribute-based authentication can support any of the trust management models mentioned; also, in a distributed system where credential verifiers do not know credential issuers (and their keys) on beforehand, credential verification conditions and rules can be modified to express more general authority delegation and trust management policies (e.g., "I accept a credential issued by an issuer satisfying trust or delegation condition Y" instead of "I accept a credential from issuer X." As the issuers in a certificate chain can be publicly known entities, also automatic certificate chain collection could be realized.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the design and implementation of the *idemix* anonymous credential system. The high-level primitives that were introduced to define the system's interfaces are easy to use and understand, and easy to extend to include new options and features. We also presented an example infrastructure for applications to exploit *idemix* authentication in an access control infrastructure.

The *idemix* system as implemented and presented here,

does not yet include features such as *all-or-nothing* non-transferability, or use for signature generation. A new NymSystem library is being implemented which will incorporate these additional features.

Deployment of *idemix* as a privacy-enhanced PKI also requires features supported by the core NymSystem, such as changing of organizations' public *idemix* keys, or for efficient revocation of credentials. We are currently developing the protocols supporting these features.

Acknowledgements

The authors are grateful to Marco Bove, Endre Bangerter, Roger Mathys, Martin Schaffer, and Dieter Sommer for their amazing Java programming making the idemix prototype reality.

8. REFERENCES

[1] R. Barrett, P. P. Maglio, and D. C. Kellem. WBI development kit. http://www.almaden.ibm.com/cs/wbi/.

[2] S. Bellovin and P. Metzger. Simple Public Key Infrastructure (SPKI) Charter. http://www.ietf.org/html.charters/spki-charter.html.

[3] M. Blaze, J. Feigenbaum, and A. D. Keromytis. Keynote: Trust management for public-key infrastructures. In *1998 Security Protocols International Workshop*, vol. 1550 of *LNCS*, pp. 59–63, 1998.

[4] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. Research in Security and Privacy, 1996. IEEE Computer Society, Technical Committee on Security and Privacy.

[5] M. Bove. Key management, setup and implementation of an anonymous credential system. Master's thesis, 2001.

[6] J. Camenisch and A. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In *EUROCRYPT 2001*, vol. 2045 of *LNCS*, pp. 93–118. Springer Verlag, 2001.

[7] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, Feb. 1981.

[8] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[9] D. Chaum and J.-H. Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In *CRYPTO '86*, vol. 263 of *LNCS*, pp. 118–167. Springer-Verlag, 1987.

[10] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO '88*, vol. 403 of *LNCS*, pp. 319–327. Springer Verlag, 1990.

[11] Consultation Committee. *X.509: The Directory Authentication Framework*. International Telephone and Telegraph, International Telecommunications Union, Geneva, 1989.

[12] I. B. Damgård. Payment systems and credential mechanism with provable security against abuse by individuals. In *CRYPTO '88*, vol. 403 of *LNCS*, pp. 328–335. Springer Verlag, 1990.

[13] C. Dwork, J. Lotspiech, and M. Naor. Digital signets: Self-enforcing protection of digital information. 1996.

[14] T. Eirich. KeyMan. http://www.alphaworks.ibm.com/tech/keyman.

[15] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. *SPKI Certificate Theory*. Internet Engineering Task Force RFC 2693.

[16] A. Fiat and A. Shamir. How to prove yourself: Practical solution to identification and signature problems. In *CRYPTO '86*, vol. 263 of *LNCS*, pp. 186–194. Springer Verlag, 1987.

[17] O. Goldreich, B. Pfitzman, and R. Rivest. Self-delegation with controlled propagation — or — what if you lose your laptop. In *CRYPTO '98*, vol. 1642 of *LNCS*, pp. 153–168, 1998. Springer Verlag.

[18] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):84–88, February 1999.

[19] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pp. 2–14, 2000. IEEE Press.

[20] N. Li, B. Grosof, and J. Feigenbaum. A practically implementable and tractable delegation logic. In *"Proceedings of the 2000 IEEE Symposium on Security and Privacy"*, pp. 27–43, 2000.

[21] N. Li, B. N. Grosof, and J. Feigenbaum. A logic-based knowledge representation for authorization with delegation. In *"Proceedings of the 12th IEEE Computer Security Foundations Workshop"*, 162–174.

[22] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *"Proceedings of the 2002 IEEE Symposium on Security and Privacy"*, pp. 114 – 130, 2002. IEEE Press.

[23] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management: extended abstract. In *8th ACM CCS*, pp. 156–165. ACM Press, 2001.

[24] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, vol. 1758 of *LNCS*, 1999.

[25] R. Mathys. New idemix client handbuch. Technical report, December 2001.

[26] A. Pfitzmann, B. Pfitzmann, and M. Waidner. Isdnmixes: Untraceable communication with very small bandwidth overhead, 1991.

[27] M. K. Reiter and A. D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[28] S. G. Stubblebine, P. F. Syverson, and D. M. Goldschlag. Unlinable serial transactions: Protocols and applications. *ACM Transactions on Information and System Security*, 2(4):354–389, Nov. 1999.