# Privacy-Preserving Credential Verification for Non-monotonic Trust Management Systems

Changyu Dong,Giovanni Russello and Naranker Dulay

Department of Computing
Imperial College London
180 Queen's Gate, London, SW7 2AZ, UK
{changyu.dong,g.russello,n.dulay}@imperial.ac.uk

**Abstract.** Trust management systems provide a flexible way for performing decentralized security management. However, most trust management systems only support monotonic policies. Compared with non-monotonic policies, monotonic ones are less flexible and cannot express policies such as "Chinese wall policies" and "separation of duties". To support non-monotonic policies, trust management systems must be able to correctly identify the credentials which a subject has that are required by the policies. Previous efforts address the problem by letting the system query the issuers directly to verify the possession status of the credentials. But this approach can violate the subject's privacy. The main contribution of this paper is a cryptographic credential verification scheme for non-monotonic trust management systems that can correctly identify the credentials that a subject has while also protecting the subject's privacy. We also analyze the security of the scheme and prove that with correct construction and certain cryptographic assumptions, the scheme is secure.

**Key words:** Trust Management, Non-monotonic Policy, Privacy, Cryptography

## 1 Introduction

In the past ten years, we have seen the emergence of trust management systems for access control and privacy protection. Trust management systems have advantages in flexibility, scalability and extensibility over traditional security mechanisms and support decentralized security management for contemporary distributed computing environments.

Trust management was first proposed by Blaze *et al.* [1]. It aims to provide a unified approach to specify and interpret security policies, credentials and relationships that allow direct authorization of security-critical actions. The basic problem that they address is: "Does the set of credentials $C$ prove that the request $R$ complies with the local security policy $P$?"

Most trust management systems, such as [2–6], assume monotonicity: additional credentials can only result in the increasing of privilege. There are several reasons why monotonicity is a desirable property in trust management [7, 4, 8]. Firstly, monotonicity simplifies the design of trust management systems. The systems do not need to evaluate all potential policies and credentials, but are still provably correct and analyzable. Monotonicity also avoids policy conflicts [9, 10] which are often caused by non-monotonicity. Furthermore, in some cases, non-monotonic policies can be converted into monotonic policies. For example, instead of defining a negative policy that requires credential $C$, one can define a positive policy to require a credential of “not have $C$”.

The monotonic assumption oversimplifies the real world by cutting off the negative part, thus it cannot handle many important scenarios. For example, with monotonic semantics, it is hard to express explicit negative policies such as a consultant cannot serve company A and B at the same time because there is a conflict of interest (the Chinese Wall policy); a bank teller should not be an auditor of the same bank (Separation of Duties). Explicit negation is particularly important for authorization in distributed system scenarios, where the number of potential requesters is high. Without negations, we cannot express policies such as “allow all except some’ elegantly.

Non-monotonicity allows more flexible and expressive security policies [11, 12, 10]. The difficulty with non-monotonic trust management systems is that the systems must have the exact set of credentials from an entity to make a sound decision. It is hard because of information asymmetry. If a subject knows or can predict that a certain credential will result in the decrease of its privileges, it may prefer not to reveal it. A trust management system cannot distinguish whether the absence of certain credentials is caused by “not having” or “not disclosing”. To solve this problem, previous studies on non-monotonic trust management [13–16] suggest that the system should be able to collect credentials directly from the credentials issuers rather than only from the subjects. Although this approach seems to be able to solve the problem, it causes new problems. One problem is privacy: the issuer could disclose information about the subject, i.e. the credential, to anyone who wants the credential. It also requires issuers to be always online, which may not be practical.

To handle non-monotonicity in trust management systems, we present a cryptographic credential verification scheme which guarantees that the trust management system can identify all the required credentials possessed by the subject while also providing protecting the subject’s privacy.

## 2 Problem Definition

Let’s consider a trust management system controlling access to a resource. Let $\mathcal{V}$ be the set of atomic privileges, $\mathcal{C}_p$ be the set of all credentials relevant to the trust decision, the trust policies can be formalized as $pol : \mathcal{P}(\mathcal{C}_p) \rightarrow \mathcal{P}(\mathcal{V})$, where $\mathcal{P}(\mathcal{C}_p)$ and $\mathcal{P}(\mathcal{V})$ are the power sets of $\mathcal{C}_p$ and $\mathcal{V}$ respectively. Loosely speaking, this means that given a set of relevant credentials, the trust management system can decide a set of privileges based on its local trust policies.

If the policies are monotonic, then we have $\mathcal{C}_1 \subseteq \mathcal{C}_2 \rightarrow pol(\mathcal{C}_1) \subseteq pol(\mathcal{C}_2)$ for all $\mathcal{C}_1$, $\mathcal{C}_2 \in \mathcal{P}(\mathcal{C}_p)$. In contrast, if the policies are non-monotonic, then there exists $\mathcal{C}_1$, $\mathcal{C}_2 \in \mathcal{P}(\mathcal{C}_p)$ such that $\mathcal{C}_1 \subseteq \mathcal{C}_2 \wedge pol(\mathcal{C}_1) \nsubseteq pol(\mathcal{C}_2)$.

One required security property of trust management systems is that the subject should not receive excessive privileges. In other words, for a subject who has a set of credentials $\mathcal{C}_s$, the privileges it can get should be bound by $pol(\mathcal{C}_r)$ where $\mathcal{C}_r = \mathcal{C}_s \cap \mathcal{C}_p$ are the credentials possessed by the subject and required by the policies (see Figure 1). It seems trivial since given $\mathcal{C}_p$, for each set of credentials $\mathcal{C}_s$, there is exactly one $\mathcal{C}_r$. However, in most cases, the system only knows $\mathcal{C}_{s'}$ which is a set of credentials collected by it. The policy evaluation is therefore based on $\mathcal{C}_{r'} = \mathcal{C}_{s'} \cap \mathcal{C}_p$ rather than $\mathcal{C}_r$. The credentials are digital assertions signed by the credential issuers and are unforgeable, i.e. $\mathcal{C}_{s'} \subseteq \mathcal{C}_s$. In consequence, it is clear that $\mathcal{C}_{r'} \subseteq \mathcal{C}_r$.

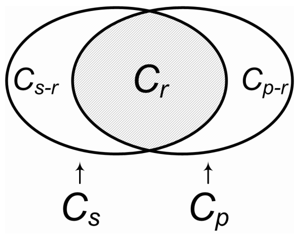

**Fig. 1.** Credential Sets.

In monotonic trust management systems, the required property is preserved in all situations. Since $C_{r'} \subseteq C_r$, by monotonicity, $pol(\mathcal{C}_{r'}) \subseteq pol(\mathcal{C}_r)$ is always true. But in non-monotonic trust management systems, the potential privileges the subject can get is bound by $\bigcup_{\mathcal{C}_{ri'} \in \mathcal{P}(\mathcal{C}_r)} pol(\mathcal{C}_{ri'})$. This means that if the system cannot identify $\mathcal{C}_r$ correctly, the subject may get more privileges than it should. As a result, credential collection and verification is crucial to non-monotonic trust management systems.

In monotonic trust management systems, credentials are usually submitted by the subjects. This is obviously not appropriate in non-monotonic trust management systems. A scheme that most non-monotonic trust management systems use is to actively collect and verify the credentials. For each credential $c_i \in \mathcal{C}_p$, the system sends a query to the credential issuer. The issuer returns a positive reply if it has issued $c_i$ to the subject, a negative reply otherwise. If the issuer's reply is positive, the system can infer that $c_i \in \mathcal{C}_s$ and in consequence, $c_i \in \mathcal{C}_r$. If the reply is negative, then $c_i \in \mathcal{C}_{p-r}$, where $\mathcal{C}_{p-r}$ is the set of credentials that were required by the policies but not possessed by the subject. After receiving definite replies for all the credentials in $\mathcal{C}_p$, the system identifies the correct set $\mathcal{C}_r$.

The scheme is problematic in the sense that it considers little about the subject's privacy. Credentials may contain sensitive information about the subject, but there is no way to prevent a malicious system from probing the credentials the subject has, e.g. the system can query about a credential in $\mathcal{C}_{s-r}$, which is not relevant to the trust decision at all. It can be even worse since the query is open to everyone. Another noticeable defect is that the query may not always get a definite reply. A query can go unanswered if the issuer is offline. In such situations, the system cannot verify the possession status of the credential.

In the following sections, we will describe a new credential verification scheme designed for non-monotonic trust management systems. The scheme allows the system to identify $\mathcal{C}_r$ efficiently and correctly. The scheme also protects the subject's privacy. The verification must first be permitted by the subject, and after the verification, the system knows nothing about the credentials in $\mathcal{C}_{s-r}$.

## 3 Credential Verification Scheme

Our credential verification scheme tries to keep a balance between avoiding unnecessary security breaches caused by lack of information and respecting the users right of controlling their information. In sections 6 and 6 we will prove that the scheme is:

- Correct. The scheme can correctly identify all the credentials that the subject has that are required by the target. And also,
- Privacy-preserving. The verification is controlled by the subject. Without the permission of the subject, the target cannot learn anything about the credentials possessed by the subject.

### 3.1 Overview of the Scheme

There are three roles in our scheme:

- Subjects: The subjects are entities who send access requests and need to be authorized.
- Targets: The targets are entities who provide resources and make the trust decisions.
- Credential issuers: Issuers create the credentials, and also credential profiles (see section 3.2) to allow the targets to identify the credentials possessed by the subjects.

As described earlier, the aim of a credential verification scheme for non-monotonic trust management systems is to identify the correct $\mathcal{C}_r$. The traditional scheme achieves the goal by finding two mutually exclusive credential sets $\mathcal{C}_r$ and $\mathcal{C}_{p-r}$ such that $\mathcal{C}_p = \mathcal{C}_r \cup \mathcal{C}_{p-r}$. This approach relies totally on the target to verify the credentials. Our approach is different. In our scheme, we let the subject provide a credential set $\mathcal{C}'_r$ such that $\mathcal{C}'_r \subseteq \mathcal{C}_p$ and $\mathcal{C}'_r \subseteq \mathcal{C}_s$. Then the subject must convince the target that $\mathcal{C}'_r = \mathcal{C}_r$ by proving $\mathcal{C}_{p-r'} \cap \mathcal{C}_{s-r'} = \emptyset$, where $\mathcal{C}_{p-r'} = \mathcal{C}_p - \mathcal{C}'_r$, $\mathcal{C}_{s-r'} = \mathcal{C}_s - \mathcal{C}'_r$.

To see that this is correct, first let's assume that when $\mathcal{C}_{p-r'} \cap \mathcal{C}_{s-r'} = \emptyset$ , $\mathcal{C}'_r \neq \mathcal{C}_r$. Because $\mathcal{C}'_r \subseteq \mathcal{C}_p$ and $\mathcal{C}'_r \subseteq \mathcal{C}_s$ and $\mathcal{C}_r = \mathcal{C}_p \cap \mathcal{C}_s$, the only possibility of $\mathcal{C}'_r \neq \mathcal{C}_r$ is $\mathcal{C}'_r \subset \mathcal{C}_r$, therefore we can find a non-empty credential set $\mathcal{C}''_r$ such that $\mathcal{C}''_r \cap \mathcal{C}'_r = \emptyset$ and $\mathcal{C}''_r \cup \mathcal{C}'_r = \mathcal{C}_r$. Then it follows that $\mathcal{C}_{p-r'} \cap \mathcal{C}_{s-r'} = \mathcal{C}''_r$, which contradicts the assumption. So $\mathcal{C}'_r = \mathcal{C}_r$ must be true.

The difficulty with our scheme is how to preserve the privacy of the subject, namely how to effectively prove $\mathcal{C}_{p-r'} \cap \mathcal{C}_{s-r'} = \emptyset$ without letting the target know any credentials in $\mathcal{C}_{s-r}$. We address the problem by constructing a cryptographic bijection mapping $\rho : \mathcal{C}_s \rightarrow \mathcal{E}_s$. $\mathcal{E}_s$ is publicly available to any entity through a highly available P2P directory service. $\rho$ is constructed using well-defined cryptographic primitives, so $\mathcal{E}_s$ discloses no information about $\mathcal{C}_s$ to the targets. The subject must prove that for any credential $c_i \in \mathcal{C}_{p-r'}$, $c_i \notin \rho^{-1}(\mathcal{E}_{s-r'})$. Because $\rho$ is a bijection, so $\rho^{-1}(\mathcal{E}_{s-r'}) = \rho^{-1}(\rho(\mathcal{C}_{s-r'})) = \mathcal{C}_{s-r'}$. Therefore the above proof is equivalent to proving $\mathcal{C}_{p-r'} \cap \mathcal{C}_{s-r'} = \emptyset$. The proof is zero-knowledge, thus at the end, the target can be convinced about the statement but knows nothing more.

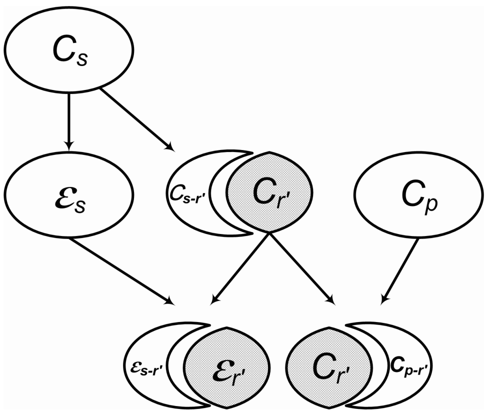

**Fig. 2.** Overview of our approach.

## 3.2 Architecture

In our scheme, $\mathcal{E}_s$ is implemented as a *credential profile* (or profile for simplicity) which allows targets to verify which credentials the subjects has. Each credential profile has four basic components. The components are:

- ID Hash: The hash value of the subject's identity. It can be used to search all the profiles associated with the identity.
- Profile Entries: Each entry is *linked* to a credential held by the subject and contains some encrypted information. The target can verify that the subject has the linked credential by performing a computation on the entry. Profile entries are discussed in more detail later.

– Timestamp: The time when this profile was created. It allows entities to determine which profile is the latest.
– Signature: The digital signature of the issuer for this profile. This signature is used to ensure that no one can modify the profile after it has been created.

The profiles are distributed independently of the credentials through a P2P directory service. The P2P directory service is maintained by the credential issuers and can be used by any entity. The advantage of the P2P approach is that each profile can be duplicated and stored in multiple places over a wide area. Therefore it provides higher availability of the profiles than storing them in one place.

To ensure that all credential information is preserved in the profile, we use an "onion" structure for the profile. If the subject has $n$ credentials, its profile has $n+1$ layers. The innermost layer of the profile, layer 0, is the ID hash of the subject. Each layer $i$, consists of a profile entry, a timestamp and is wrapped by an overall signature on the content of layer 0 to layer $i$. The onion structure is built up along with the updating process of the subject's credential set. As shown in Figure 3, each time the subject needs a credential, it contacts the credential issuer (1). The credential issuer generates a credential for the subject, at the same time it must also create a new profile for the subject. To do so, the credential issuer first needs to obtain the latest version of the subject's profile. This can be done by searching the P2P directory service using the hash value of the subject's identity (2). After getting the latest profile (3), the issuer generates a new entry for the credential it is issuing to the subject and also a timestamp, then appends them to the old profile. The issuer then signs the new profile and releases it to the subject (4) with the credential and also to the P2P directory service (5). As we can see, by using the onion structure, we ensure that the next time that a credential issuer creates a profile, it cannot modify or remove any content from the old profile. Suppose it modifies the content in layer $k$, it would then need to forge all the signatures from layer $k$ to layer $n$. We also require the peers in the P2P directory service to check the contents of a newer versions of a profile with their local version, and reject them if the checking fails.

### 3.3 Cryptographic Building Blocks

Our scheme is realised by using cryptographic techniques. In this section, we briefly outline the cryptographic primitives used. In section 4, we will show how the security of our scheme follows the security properties of the primitives. For more information about these cryptographic primitives and their formal security definition, please refer to [17–19]. The cryptographic primitives used are:

1. *A commitment scheme*, $Commit : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^l$. It is a two-phase protocol between two parties, the committer and the receiver. In the first phase (the commitment phase), the committer commits to $r \in \{0,1\}^n$ by choosing a Secret $s \in \{0,1\}^k$ to generate a commitment $Commit_s(r)$ through a polynomial time algorithm which binds $r$ to $Commit_s(r)$, i.e. it's

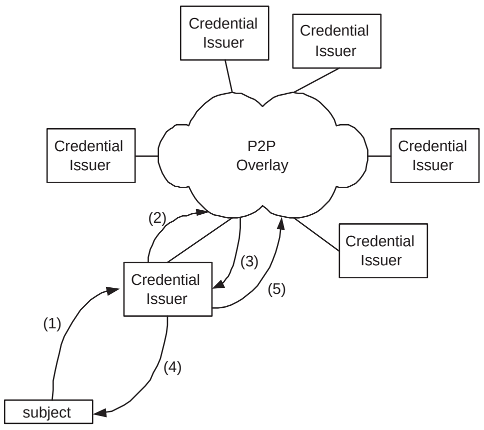

**Fig. 3.** Issuing a credential using the P2P directory service.

infeasible for the receiver to find $r'$ and $s'$ which produce the same commitment $Commit_{s'}(r') = Commit_s(r)$ (this is the *binding property*). The committer sends $Commit_s(r)$ to the receiver. Given only the commitment, it's infeasible for the receiver to compute the committed string $r$ (this is the *hiding property*). In the second phase (the open phase), the committer reveals $r$ and $s$ to the receiver. Now the receiver checks whether they are valid against the commitment, if the receiver can compute $Commit_s(r)$ from $r$ and $s$, then it is convinced that $r$ was indeed committed by the committer in the first phase, otherwise it rejects $r$.

2. *Zero-Knowledge Proof Protocols.* Let $P$, $V$ be two Interactive Turing Machines, $L$ be a language over $\{0,1\}^*$, the goal of a zero-knowledge proof protocol $(P,V)$ is to allow the prover $P$ to prove to the verifier $V$ that a given $x$ belongs to language $L$, without disclosing any other information.
   In the following sections, we will use the notion introduced in [20] for the zero-knowledge proofs. The convention is that the elements listed in the round brackets before ":" denote the knowledge to be proved to the verifier and all other parameters are known to the verifier. For example: $PK\{(a,b): y = g^a h^b\}$ means a zero-knowledge proof of integers $a, b$ such that $y = g^a h^b$ holds and $y, g, h$ are known to the verifier.

## 3.4 Profile Entry

Profile entries (or entries for simplicity) can be used by the target to verify that the subject possesses the corresponding credential. Entries are generated by using cryptographic techniques so that one cannot learn anything about the credentials from the entries, unless following the credential verification protocol.

We assume that there is a common vocabulary for specifying credentials which is used by all the entities in the system. We also assume that each credential has a credential name, e.g. student, member etc.. An entry is linked to a credential whose name is $c$ and generated by the credential issuer when it issues the credential. To generate an entry, the creator (the credential issuer) first creates a commitment for the credential name $Commit_s(c)$. The secret $s$ for opening the commitment will be sent to the subject through a secure channel. The entry is a tuple $(Commit_s(c), Sig(cred), exp\_time)$. $Sig(cred)$ is the signature of the linked credential and is used to associate the entry with the real credential. $exp\_time$ is the expiration time of the credential. An entry can be revoked implicitly or explicitly. Each entry contains the expiration time of the linked credential. The entry becomes invalid when the credential expires. When a credential is revoked before expiring, the credential issuer puts the signature of the credential into a revocation list, and publishes it into the P2P network.

We use a modified Pedersen Commitment Scheme which is slightly different from the standard one.

**Setup** The issuer chooses two large prime numbers $p$ and $q$ such that $q$ divides $p-1$. Let $g$ be a generator of $G_q$, the unique order-$q$ subgroup of $\mathbb{Z}_p^*$. The issuer chooses $x$ uniformly randomly from $\mathbb{Z}_q$ and computes $h = (g^x \bmod p)$. The issuer keeps the value $x$ secret and makes the values $p, q, g, h$ public.

**Commit** The issuer chooses $s$ uniformly randomly from $\mathbb{Z}_q$ and computes the commitment $Commit_s(c) = g^c h^s$.

There are three parties involved: a committer (the issuer), a prover (the subject) and a receiver(the target). The issuer generates the commitment to $c$ and lets the subject know the secret for opening the commitment. This is because the binding property can only guarantee that after generating the commitment, the committer cannot change what it committed to; however, it provides no guarantee on what can be committed to. If we let the subject generate the commitment, it could commit to another credential name $c'$ rather than $c$ and it could take advantage from this, i.e. to hide the credential in order to gain excessive privileges. So we let a trusted third party (the issuer) generate the commitment to ensure that the commitment is indeed a commitment to $c$. Note that the subject does not need to open the commitment. What the subject needs to do is to use the commitment and the secret $s$ to convince the target by a zero-knowledge proof protocol that the linked credential is not required.

## 3.5 Credential Verification Protocol

The protocol is described in the following and shown in a message sequence chart in Figure 4. Note: any party can terminate the process if a malicious behavior is detected during the protocol. If the protocol terminates prematurely, it will output "$\perp$".

1. The target receives a request from the subject.
2. The target decides the credentials that need to be checked according to its local policies, i.e. $\mathcal{C}_p$, and lets the subject know $\mathcal{C}_p$.

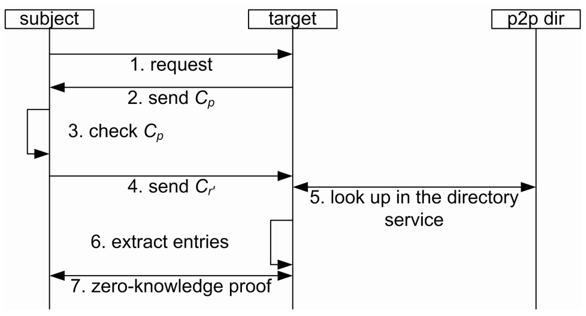

**Fig. 4.** Message sequence chart of the protocol for credential verification.

3. The subject decides $\mathcal{C}_r = \mathcal{C}_p \cap \mathcal{C}_s$. If there is any credential in $\mathcal{C}_r$ which is sensitive and the subject does not want the target to know, it can choose to refuse and terminate the process. Otherwise, the subject proceeds. Note: as we have mentioned before, $\mathcal{C}_p$ is the set of all the credentials the target needs to check according to its local policies, not the the credentials the subject must have, so even if the subject does not have all the credentials requested, it can still choose to proceed.
4. If the subject chooses to proceed, it sends $\mathcal{C}'_r = \mathcal{C}_r$ to the target. The target checks the validity of the credentials in $\mathcal{C}'_r$. If $\mathcal{C}'_r$ contains all the credentials that appear in $\mathcal{C}_p$, i.e. $\mathcal{C}'_r = \mathcal{C}_p$, the protocol terminates and outputs $\mathcal{C}'_r$, otherwise the target creates the set $\mathcal{C}_{p-r'} = \mathcal{C}_p - \mathcal{C}'_r$ and the protocol continues.
5. The target obtains the subject's latest credential profile $P$ from the P2P directory service.
6. The target creates a set containing all the valid entries extracted from $P$. Valid means not expired or revoked. The set is effectively equal to $\mathcal{E}_s$. For each valid entry $(Commit_{s_j}(c_j), Sig(cred_j), exp\_time_j)$, if the target can find a credential in $\mathcal{C}'_r$ whose signature is $Sig(cred_j)$, then this entry is removed from $\mathcal{E}_s$. At the end, the target will have the set $\mathcal{E}_{s-r'}$.
7. For each entry $(Commit_{s_k}(c_k)$, $Sig(cred_k)$, $exp\_time_k)$ in $\mathcal{E}_{s-r'}$, the subject must run a zero-knowledge proof protocol as described in section 3.6 to convince the target that there is no credential in $\mathcal{C}_{p-r'}$ whose name is $c_k$. The target will then know that the credential is not required by its policies, but nothing more than that.
8. The credential verification protocol completes by outputting $\mathcal{C}'_r$.

### 3.6 Zero-Knowledge Proof Protocol

The zero-knowledge proof protocol that we use in step 7 above is adapted from [21]. The original protocol is a two-party secure computation protocol used to compare two integers. We simplify the settings because there is only one secure input, which is the name of the credential held by the subject.

The aim of this protocol is: given public security parameters $p, q, g, h$, a commitment $g^c h^s$ as described in 3.4, and a different credential name $c' \in \mathbb{Z}_q$, the subject must prove $c' \neq c$ to the target.

We use two well-defined sub-protocols in the proof. The first one is Schnorr's protocol [22] $PK\{(x) : y = g^x\}$ which proves knowledge of a discrete logarithm. The other is Okamoto's protocol [23] $PK\{(a, b) : y = g_1^a g_2^b\}$ which proves knowledge of how to open a commitment. It can be easily extended to $PK\{(a, b) : (x = g_1^a) \wedge (y = g_2^a g_3^b)\}$. The zero-knowledge proof protocol is:

1. The subject uniformly randomly chooses $x \in \mathbb{Z}_q$, $x$ is kept secret. It computes $h_2 = h^x$ and sends it to the target. Then it proves to the target that it knows $x$ by $PK\{(x) : h_2 = h^x\}$.
2. The subject computes $P_s = h_2^s$ where $s$ is the secret to open the commitment $Commit_s(c) = g^c h^s$ and sends it to the target. Then it proves to the target that it knows $c, s$ and that it used the same $s$ in computing $h_2^s$ as in computing $g^c h^s$ by $PK\{(c, s) : (P_s = h_2^s) \wedge (Commit_s(c) = g^c h^s)\}$.
3. The target also selects a random element $s' \in \mathbb{Z}_q$ and computes $P_{s'} = h_2^{s'}$ and $Commit_{s'}(c') = g^{c'} h^{s'}$. The target sends $Commit_{s'}(c')$ to the subject as a challenge.
4. The subject computes $Q = (\frac{Commit_s(c)}{Commit_{s'}(c')})^x$, sends the result to the target, and proves $PK\{(x) : h_2 = h^x \wedge Q = (\frac{Commit_s(c)}{Commit_{s'}(c')})^x\}$.
5. Finally, the target checks whether $Q \neq \frac{P_s}{P_{s'}}$ holds. If so, then $c' \neq c$

We have implemented a proof-of-concept prototype in Java 1.5 and done a preliminary performance evaluation. The evaluation was done on a Pentium IV 3.2 GHz (dual core) desktop with 1 GB memory. The execution of the zero-knowledge protocol takes about 130 milliseconds. The result was obtained by averaging the time for 1000 executions. The performance can be further improved by optimizing the code.

## 4 Security Analysis

In our credential verification protocol, the security requirements of the parties are different. For the subject, the requirement is for the scheme to prevent a malicious target from learning excessive information about its credentials. For the target, the requirement is for the scheme to correctly identify the subject's credentials and prevent a malicious subject from cheating about the credentials it has.

To reflect the requirements of both parties, we define the security of our protocol as follows:

1. Correctness. Given the target follows the protocol, if the subject sends $\mathcal{C}'_r = \mathcal{C}_r$ to the target, then at the end, the protocol should output $\mathcal{C}_r$ with overwhelming probability; otherwise, the protocol should output $\perp$ with overwhelming probability.

2. Privacy-preserving. Given the subject follows the protocol, the target should learn either nothing or the set of credentials $\mathcal{C}_r$. In the latter case, it should be computationally infeasible for a malicious target to learn any credential in $\mathcal{C}_{s-r}$.

The proofs can be found in the appendix.

## 5 Related work

There are few non-monotonic trust management systems. REFEREE [13] is a trust management system for web applications. It uses PICS labels [24] as credentials and assumes they can be obtained from authorities' websites. The system is responsible for collecting all the credentials, therefore it is possible to gather complete information. TPL [15] allows negative credentials which are interpreted as suggestions of "not to trust". In TPL, positive credentials are submitted by the subject, and the negative credentials are collected by the system. [25] discusses non-monotonic access policies in trust negotiation and argues that to avoid relying on outside information, the system should only have non-monotonic policies according to its local information and the credentials submitted to the system should be monotonic. A recent study [16] adds a restricted form of negation to the standard RT trust management language. But as we have mentioned in section 1, none of these systems address privacy or availability issues.

## 6 Conclusion and Future Work

In this paper we discussed the benefits and problems of non-monotonic trust management systems. To handle non-monotonicity, we developed a credential verification scheme which guarantees that the system can identify all the required credentials possessed by the subject while also protecting the subject's privacy. The scheme is implemented by using several cryptographic primitives. We also analyzed our scheme and proved that with correct construction and certain cryptographic assumptions, our scheme is secure.

One aspect of our future work is to allow more expressive trust policies. Currently, our scheme does not support wildcard credential names in policies, for example, the policy "a subject can access the patient's medical record if it has a doctor credential signed by any hospital". At present, such policies cannot be handled directly. The example has to be translated into verifying the subject has at least one credential in the list of all doctor credentials signed by a hospital. This approach is static and also increases the computation time because in the worst case, all the credential names that appear in the list need to be verified.

We will also investigate using the cryptographic credential verification scheme in automated trust negotiation [26]. Automated trust negotiation is a promising approach to build trust management systems in a privacy-preserving way. Disclosure policies are established to regulate the disclosure of sensitive credentials and policies. Traditional trust negotiation systems disclose the credentials incrementally, therefore must be monotonic. They are also subject to policy cycles where a negotiator $A$ has a disclosure policy that requires credential $c_1$ from another negotiator $B$ before disclosing credential $c_2$ while $B$ has a disclosure policy that requires credential $c_2$ from $A$ before disclosing credential $c_1$. When there is a policy cycle, the negotiation fails. In [27], the authors propose the *Reverse Eager (RE) trust negotiation strategy* in which two negotiators start from the maximum credentials sets and in each iteration prune the credentials sets by removing the unusable credentials according to their own policies. The RE strategy is cycle-tolerant which means even with policy cycles, the negotiation can still succeed. But the RE strategy does not support non-monotonic policies and the trust negotiation protocol requires intensive computation. We are looking at developing a more efficient protocol using our credential verification scheme and the RE strategy for non-monotonic and cycle-tolerant trust negotiation.

**Acknowledgments.** This research was supported by the UK's EPSRC research grant EP/C537181/1 and forms part of CareGrid, a collaborative project with the University of Cambridge. The authors would like to thank the members of the Policy Research Group at Imperial College for their support.

## References

1. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (1996) 164–173
2. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: Rfc2704: The keynote trust-management system version 2 (1999)
3. Jim, T.: Sd3: A trust management system with certified evaluation. In: SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (2001) 106–115
4. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a role-based trust-management framework. In: SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (2002) 114–130
5. Hess, A., Seamons, K.E.: An access control model for dynamic client-side content. In: SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies, New York, NY, USA, ACM Press (2003) 207–216
6. Carbone, M., Nielsen, M., Sassone, V.: A formal model for trust in dynamic networks. In: SEFM, IEEE Computer Society (2003) 54–61
7. Blaze, M., Feigenbaum, J., Strauss, M.: Compliance checking in the policymaker trust management system. In: FC '98: Proceedings of the Second International Conference on Financial Cryptography, London, UK, Springer-Verlag (1998) 254–274
8. Seamons, K., Winslett, M., Yu, T., Smith, B., Child, E., Jacobson, J., Mills, H., Yu, L.: Requirements for policy languages for trust negotiation. In: POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02), Washington, DC, USA, IEEE Computer Society (2002) 68–79

9. Lupu, E.C., Sloman, M.: Conflicts in policy-based distributed systems management. IEEE Trans. Softw. Eng. **25**(6) (1999) 852–869
10. Jajodia, S., Samarati, P., Subrahmanian, V.S., Bertino, E.: A unified framework for enforcing multiple access control policies. In: SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM Press (1997) 474–485
11. Clark, D.D., Wilson, D.R.: A comparison of commercial and military computer security policies. In: IEEE Symposium on Security and Privacy. (1987) 184–195
12. Brewer, D.F.C., Nash, M.J.: The chinese wall security policy. In: IEEE Symposium on Security and Privacy. (1989) 206–214
13. Chu, Y.H., Feigenbaum, J., LaMacchia, B., Resnick, P., Strauss, M.: Referee: trust management for web applications. Comput. Netw. ISDN Syst. **29**(8-13) (1997) 953–964 283252.
14. Li, N., Feigenbaum, J., Grosof, B.N.: A logic-based knowledge representation for authorization with delegation (extended abstract). In: Proceedings of the 1999 IEEE Computer Security Foundations Workshop, IEEE Computer Society Press (June 1999) 162–174
15. Herzberg, A., Mass, Y., Mihaeli, J., Naor, D., Ravid, Y.: Access control meets public key infrastructure, or: assigning roles to strangers. In: the 2000 IEEE Symposium on Security and Privacy, Berkeley, CA (2000) 2–14
16. Czenko, M., Tran, H., Doumen, J., Etalle, S., Hartel, P., den Hartog, J.: Nonmonotonic trust management for p2p applications. Electronic Notes in Theoretical Computer Science **157**(3) (2006) 113–130
17. Goldreich, O.: Foundations of Cryptography: Volume I Basic Tools. Cambridge University Press (2001)
18. Goldwasser, S., Bellare, M.: Lecture notes on cryptography. http://www-cse.ucsd.edu/users/mihir/papers/gb.pdf
19. Goldreich, O.: Foundations of Cryptography: Volume II Basic Applications. Cambridge University Press (2004)
20. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In Jr., B.S.K., ed.: CRYPTO. Volume 1294 of Lecture Notes in Computer Science., Springer (1997) 410–424
21. Boudot, F., Schoenmakers, B., Traoré, J.: A fair and efficient solution to the socialist millionaires' problem. Discrete Applied Mathematics **111**(1-2) (2001) 23–36
22. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptology **4**(3) (1991) 161–174
23. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, London, UK, Springer-Verlag (1993) 31–53
24. Resnick, P., Miller, J.: Pics: Internet access controls without censorship. Commun. ACM **39**(10) (1996) 87–93
25. Dung, P.M., Thang, P.M.: Trust negotiation with nonmonotonic access policies. In Aagesen, F.A., Anutariya, C., Wuwongse, V., eds.: INTELLCOMM. Volume 3283 of Lecture Notes in Computer Science., Springer (2004) 70–84
26. Winsborough, W.H., Seamons, K.E., Jones, V.E.: Automated trust negotiation. In: DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Volume 1., Hilton Head, SC, IEEE Press (2000) 88–102
27. Frikken, K.B., Li, J., Atallah, M.J.: Trust negotiation with hidden credentials, hidden policies, and policy cycles. In: NDSS, The Internet Society (2006)

28. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In Feigenbaum, J., ed.: CRYPTO. Volume 576 of Lecture Notes in Computer Science., Springer (1991) 129–140

## Appendix

### Security Proof of the Zero-knowledge Proof Protocol

**Lemma 1.** *The protocol in section 3.6 is complete: if $c' \neq c$, then $Pr[(P,V)(c' \neq c) = 1] = 1$.*

*Proof.* The zero-knowledge proof convinces the verifier by comparing $Q = (\frac{Commit_s(c)}{Commit_{s'}(c')})^x = (\frac{g^c h^s}{g^{c'} h^{s'}})^x = g^{(c-c')x} h^{(s-s')x}$ and $\frac{P_s}{P_{s'}} = \frac{h^{sx}}{h^{s'x}} = h^{(s-s')x}$ where $c$ is a credential name and $c'$ is another credential name. $Q \neq \frac{P_s}{P_{s'}}$ holds only when $c' \neq c$, therefore the protocol is complete.

**Lemma 2.** *The protocol in section 3.6 is sound: if $c' = c$, then $\forall P'$ $Pr[(P',V)(c' \neq c) = 1] \leq \delta$, where $\delta$ is a negligible probability.*

*Proof.* If a malicious prover can manipulate $Q, P_s, P_{s'}$, then it can control the result of the zero-knowledge proof protocol. For example, if the prover can construct $P_{s'} = h^{s''x}$ using $s'' \neq s'$, then $Q \neq \frac{P_s}{P_{s'}}$ even $c = c'$. But a cheating prover can succeed with only a negligible probability. Firstly, $h^x$ is revealed to the verifier and proved to be constructed correctly in step 1 using Schnorr's protocol. In step 2, the prover must prove it uses the same $s$ in computing $P_s = (h^x)^s$ as in computing $g^c h^s$ using the extended Okamoto protocol. $Commit_s(c)$ is known by the target and $Commit_{s'}(c')$ is computed by the target, and in step 5, the prover must prove that it uses the same $x$ in computing $h^x$ and $Q = (\frac{Commit_s(c)}{Commit_{s'}(c')})^x$ using Schnorr's protocol. To manipulate $h^x$, $P_s$ and $Q$, a malicious prover must break Schnorr's protocol or the extended Okamoto protocol. But in the two sub-protocols, the challenges are chosen randomly from $[1, 2^t]$, so the probability of successful cheating is at most $2^{-t}$. When $t$ is sufficiently large, the probability is negligible. Therefore $h^x$, $Q$ and $P_s$ must be constructed correctly with a overwhelming probability. The prover cannot manipulate $P_{s'} = (h^x)^{s'}$ because $s'$ is selected by the verifier. So the protocol is sound.

**Lemma 3.** *Under the Discrete Logarithm Assumption and the Decisional Diffie-Hellman Assumption, the protocol in section 3.6 is zero-knowledge.*

*Proof.* The execution of the protocol produces a view in the form $\{h_2 = h^x, P_s = h_2^s, s', Commit_{s'}(c'), Q = (\frac{Commit_s(c)}{Commit_{s'}(c')})^x\}$. Following the definition of zero-knowledge, we need to show that there exists a probabilistic polynomial time simulator $M^*$ which can produce a simulation of a view. Note that because the simulator of the main protocol can call the simulators of the sub-protocols, and because the sub-protocols have been proven to be zero-knowledge, we omit views of the sub-protocols here.

We can construct $M^*$ as follows:

1. The public input is $p, q, g, h, g^c h^s, c'$.
2. $M^*$ randomly chooses $x^* \in \mathbb{Z}_q$, and compute $h_2^* = h^{x^*}$.
3. $M^*$ randomly chooses $s'^* \in \mathbb{Z}_q$, and computes $P_{s'}^* = (h_2^*)^{s'^*}$ and $Commit_{s'}(c')^* = g^{c'} h^{s'^*}$.
4. $M^*$ computes $Q^* = (\frac{Commit_s(c)}{Commit_{s'}(c')^*})^{x^*}$.
5. $M^*$ chooses $s^*$ such that $(h_2^*)^{s^*} \neq Q^* P_{s'}^*$, then let $P_s^* = (h_2^*)^{s^*}$.
6. $M^*$ outputs $\{h_2^*, P_s^*, s'^*, Commit_{s'}(c')^*, Q^*\}$

It is easy to see that under the Discrete Logarithm Assumption and the Decisional Diffie-Hellman Assumption, the simulation is computationally indistinguishable from a view produced in the protocol. So our protocol is zero-knowledge.

**Proof of Correctness of Credential Verification Protocol**

**Theorem 1.** *The credential verification protocol is correct.*

*Proof.* Firstly we need to prove that in the protocol, if the subject sends $\mathcal{C}_r' = \mathcal{C}_r$ to the target, then at the end, the protocol should output $\mathcal{C}_r$ with overwhelming probability. It is clear that in step 4, if an honest subject sends $\mathcal{C}_r' = \mathcal{C}_r$ to the target, then there are two cases to consider:

**Case 1:** $\mathcal{C}_r = \mathcal{C}_p$. In this case, the target will detect $\mathcal{C}_{r'} = \mathcal{C}_p$, and will terminate the protocol and output $\mathcal{C}_{r'}$, which equals $\mathcal{C}_r$.

**Case 2** $\mathcal{C}_r \subset \mathcal{C}_p$. In this case, the target has $\mathcal{C}_{p-r'} = \mathcal{C}_p - \mathcal{C}_{r'} = \mathcal{C}_p - \mathcal{C}_r = \mathcal{C}_{p-r}$ in step 4. In step 6 the target will have $\mathcal{E}_{s-r'} = \rho(\mathcal{C}_s - \mathcal{C}_{r'}) = \rho(\mathcal{C}_s - \mathcal{C}_r) = \rho(\mathcal{C}_{s-r})$. We have proven that the zero-knowledge proof protocol is complete in Lemma 1, therefore in step 7, the subject can prove that for each entry ($Commit_{s_k}(c_k)$, $Sig(cred_k)$, $exp\_time_k$) in $\mathcal{E}_{s-r'}$, that there is no credential in $\mathcal{C}_{p-r'}$ whose name is $c_k$. Then in step 8, the protocol outputs $\mathcal{C}_{r'}$, which equals $\mathcal{C}_r$.

Next we will prove that if the subject sends $\mathcal{C}_r' \neq \mathcal{C}_r$ to the target, then at the end, the protocol should output $\perp$ with overwhelming probability.

In step 4, if a malicious subject sends $\mathcal{C}_r' \neq \mathcal{C}_r$ to the target, there are also two cases to consider:

**Case 1:** $\mathcal{C}_{r'} \nsubseteq \mathcal{C}_r$. In this case, there exists at least one credential $c$ such that $c \in \mathcal{C}_{r'}$ and $c \in \mathcal{C}_{s-r}$. If $\mathcal{C}_{r'} = \mathcal{C}_r$, then all the credentials in $\mathcal{C}_{r'}$ must also be in $\mathcal{C}_p$, but now $c$ is not in $\mathcal{C}_p$ because it is in $\mathcal{C}_{s-r}$. Therefore the target can detect $\mathcal{C}_r' \neq \mathcal{C}_r$ easily. The target will then terminate the protocol and output $\perp$.

**Case 2:** $\mathcal{C}_{r'} \subset \mathcal{C}_r$. In this case, there exists a non-empty credential set $\mathcal{C}_r''$ such that $\mathcal{C}_r'' \cap \mathcal{C}_r' = \emptyset$ and $\mathcal{C}_r'' \cup \mathcal{C}_r' = \mathcal{C}_r$. In step 4, the target has $\mathcal{C}_{p-r'} = \mathcal{C}_{p-r} \cup \mathcal{C}_r''$ and in step 6, the resulting set $\mathcal{E}_{s-r'} = \rho(\mathcal{C}_{s-r} \cup \mathcal{C}_r'')$. Because $\rho^{-1}(\mathcal{E}_{s-r'}) \cap \mathcal{C}_{p-r'} = \mathcal{C}_r''$, for a credential in $\mathcal{C}_r''$ whose name is $c_k$, its entry ($Commit_{s_k}(c_k)$, $Sig(cred_k)$, $exp\_time_k$) can be found in $\mathcal{E}_{s-r'}$. We have proven that the the zero-knowledge proof protocol is sound in Lemma 2, therefore in step 7, the subject cannot convince the target. The target will terminate the protocol and output $\perp$ with overwhelming probability.

**Proof of Privacy-preservation of Credential Verification Protocol**

Now we prove that the subject's privacy is preserved by the protocol. We first need to show that the credential profile is privacy-preserving.

**Lemma 4.** *Given only the credential profile, the target learns nothing about the credential possessed by the subject.*

*Proof.* A credential profile contains profile entries. Let's look at the structure of a profile entry. Each entry is a tuple $(Commit_s(c)$, $Sig(cred)$, $exp\_time)$. $Sig(cred)$ discloses no information because the signature is generated from the hash value of the credential content. As the hash function is not invertible, no one can learn anything about the credential by looking at the signature. The commitment scheme we use is unconditionally hiding, which means even with unbounded computational power, the possibility of an adversary finding the committed value $c$ is still negligible because for any $c \in \mathbb{Z}_q$ and uniformly randomly chosen $s \in \mathbb{Z}_q$, $Commit_s(c)$ is uniformly distributed in $G_q$ [28]. So by looking at the commitment, one can learn nothing about $c$. It is impossible to infer the credential from $exp\_time$. Overall, the profile entries leaks no information about the credential.

For the other parts in the profile: an adversary cannot learn any information about the credentials from the signatures, and ID hash and timestamps contain no information about the credentials.

Therefore, given only the credential profile, the target learns nothing about the credentials possessed by the subject.

**Theorem 2.** *The credential verification protocol is privacy-preserving.*

*Proof.* The credential profile is publically available in the P2P directory service before entering the protocol, so is available to the target. But by Lemma 4, the target learns nothing about the subject's credentials by looking at the profile.

In steps 1-3, the subject discloses no additional information about its credentials, so the target still knows nothing. If the subject terminates in step 3, then the target knows nothing about the credentials in $\mathcal{C}_s$.

If the subject decides to proceed, then in step 4, the subject discloses $\mathcal{C}_r$ to the target. The target knows all the credentials in $\mathcal{C}_r$, but nothing about the credentials in $\mathcal{C}_{s-r}$. In step 7, the subject needs to run the protocol described in section 3.6 with the target. We have shown that under the Discrete Logarithm Assumption and the Decisional Diffie-Hellman Assumption, that the protocol is zero-knowledge in Lemma 3, so for any credential $c \in \mathcal{C}_{s-r}$, it is computationally infeasible for the target to learn any information other than the fact that $c \notin \mathcal{C}_r$.