Verifying Anonymous Credential Systems in Applied Pi Calculus

Xiangxi Li[1], Yu Zhang[2], and Yuxin Deng[1]⋆

[1] Department of Computer Science and Engineering, Shanghai Jiao Tong University
Shanghai, China
[2] Laboratory for Computer Science, Institute of Software, Chinese Academy of Sciences
Beijing, China

Abstract. Anonymous credentials are widely used to certify properties of a credential owner or to support the owner to demand valuable services, while hiding the user's identity at the same time. A credential system (a.k.a. pseudonym system) usually consists of multiple interactive procedures between users and organizations, including generating pseudonyms, issuing credentials and verifying credentials, which are required to meet various security properties. We propose a general symbolic model (based on the applied pi calculus) for anonymous credential systems and give formal definitions of a few important security properties, including pseudonym and credential unforgeability, credential safety, pseudonym untraceability. We specialize the general formalization and apply it to the verification of a concrete anonymous credential system proposed by Camenisch and Lysyanskaya. The analysis is done automatically with the tool ProVerif and several security properties have been verified.

1 Introduction

The use of anonymous credential systems (sometimes called pseudonym systems) [11] is by far the best known idea to protect personal information in communications. These systems use pseudonyms generated by special random processes instead of users' private information to identify the user in order to guarantee the anonymity of users. A credential can be issued to a pseudonym, and the corresponding user can show her possession of the credential, without revealing any information beyond the bare fact that she owns such a credential. For a credential system to be useful, some basic properties must be satisfied. For example, a user should not be able to make transactions with others by using a credential not issued by some valid organization (*credential unforgeability*), and transactions carried out by the same user cannot be linked (*unlinkability* or *untraceability*). In some applications it might be desirable that a credential can only be used once (*one-show credential*) or a user cannot lend her credentials to others (*non-transferability*).

It is widely known that designing good security protocols is an error-prone task. There were protocols which had been used in practical applications for many years but

⋆ The first and the third authors are supported by the National Natural Science Foundation of China (Grant No. 60703033).

later on were found to be flawed. Examples include Needham-Schroeder [14], SSL [19] and PKCS [20]. Formal methods were introduced as a promising technique to analyze security protocols and in many cases the analysis can be done with automatic tools. As a case study in this respect, we formalize the anonymous credential system proposed by Camenisch and Lysyanskaya [10] (which we shall refer to as the CL system in the sequel) in the applied pi calculus [3] and we employ the tool ProVerif [8] to analyze several security properties of the system.

To our knowledge, this is the first formal, automated verification (at the symbolic level) of this type of security systems, although many credential systems, as a main research concern of cryptography, have been verified using traditional, semi-formal approaches in cryptography. The main part of our work is devoted to the mechanized analysis of the CL system, which is probably the most complex pseudonym system targeting many security requirements but remaining efficient. We have checked that the system satisfies two very basic properties: unforgeability (of both pseudonyms and credentials) and user privacy (pseudonym untraceability). A less arresting property, which we call *credential safety* and aims at preventing unauthorized use (or stealing) of credentials, is not met by the system — the traditional replay attack breaks safety.

However, we regard our contribution as more than just a case study using ProVerif. As credential systems become more and more widely used in large-scale security applications and many protocols have been proposed, we have been very careful in formalizing the system to make our model scalable. In particular, we provide a general modeling framework in the applied pi calculus and we believe that the formalization of most credential systems falls into it. In fact, we are currently studying other systems with significantly different implementation from the CL system.

The rest of the paper is structured as follows: In Section 2 we briefly introduce the applied pi calculus, as well as the formalization of zero-knowledge proofs by Backes *et al*. Section 3 gives a general description of credential systems and an overall modeling structure in applied pi. The next section formalizes four most important security properties: pseudonym unforgeability, credential unforgeability, credential safety and pseudonym untraceability, and summarizes the verification results in ProVerif of the basic credential system of Camenisch and Lysyanskaya. Section 5 discusses related work and Section 6 concludes the paper.

2 The applied pi calculus

2.1 Syntax and semantics

We briefly recall the syntax and operational semantics of the applied pi calculus; more details can be found in [3].

A *signature* Σ is a finite set of function symbols. Given a signature Σ, an infinite set of *terms* is defined by the following grammar:

$$\begin{array}{rll} M, N := & a, b, c, \ldots, k, \ldots & \text{names} \\ \mid & x, y, z & \text{variables} \\ \mid & f(M_1, \ldots, M_l) & \text{function applications} \end{array}$$

where f ranges over the functions in Σ and l matches the arity of f. Terms are equipped with an *equational theory* E which consists of a set of equations over terms. We write $\Sigma \vdash M = N$ when the equation $M = N$ is in the theory associated with Σ, and $\Sigma \nvdash M = N$ for the opposite.

The grammar for *plain process* is similar to the one in the pi calculus [16], except that here messages can contain terms rather than names.

$$
\begin{array}{rll}
P, Q, R \ := & 0 & \text{null processes} \\
| & P|Q & \text{parallel composition} \\
| & !P & \text{replication} \\
| & \nu n \,.\, P & \text{name restriction} \\
| & \texttt{if}\ M = N\ \texttt{then}\ P\ \texttt{else}\ Q & \text{conditional} \\
| & \mathsf{u}(x)\,.\,P & \text{message input} \\
| & \overline{\mathsf{u}}(N)\,.\,P & \text{message output}
\end{array}
$$

The null process 0 does nothing and is usually omitted from process specifications. The process $P|Q$ executes P and Q in parallel, and $!P$ stands for an infinite copies of P running in parallel. The process $\nu n.P$ generates a fresh name n and behaves as P. The process $\texttt{if}\ M = N\ \texttt{then}\ P\ \texttt{else}\ Q$ behaves as P if $\Sigma \vdash M = N$, and as Q otherwise. The input process $\mathsf{u}(x).P$ can receive a message N from channel a and behaves as $P\{N/x\}$. We often take the abbreviation $\nu(\widetilde{\mathsf{u}}).P$ for $\nu\mathsf{u}_1.\cdots.\nu\mathsf{u}_n.P$ and $\mathsf{u}(= M).P$ for

$$\mathsf{u}(x)\,.\,\texttt{if}\ x = M\ \texttt{then}\ P\ \texttt{else}\ 0.$$

The output process $\overline{\mathsf{u}}\langle N\rangle.P$ sends message N on channel a and behaves as P.

Extended processes are defined with *active substitutions*:

$$
\begin{array}{rll}
A, B, C ::= & P & \text{plain process} \\
| & A|B & \text{parallel composition} \\
| & \nu x.A & \text{variable restriction} \\
| & \{M/x\} & \text{active substitution} \\
| & \texttt{event}(x_1, \ldots, x_n) & \text{events}
\end{array}
$$

where $\{M/x\}$ is the substitution that replaces the variable x with the term M. The process $\nu x.(\{M/x\}|P)$ restricts the scope of substitution in P and is often written as *let* $x =$ M *in* P. As usual, names and variables have scopes, which are delimited by restrictions and inputs. We write $fv(A)$ and $bv(A)$ (resp. $fn(A)$ and $bn(A)$) for the sets of free and bound variables (resp. names) of A. An extended process is *closed* when every variable is either bound or defined by an active substitution. Events are supported by ProVerif and are used to define traces of processes.

Every extended process can be mapped to a *frame* $\varphi(A)$ by replacing every plain process embedded in A with 0. Thus, a frame is built up from 0 and active substitutions by parallel composition and restriction. The frame $\varphi(A)$ can be viewed as the static knowledge exposed by A to the environment, but not as $A's$ dynamic behavior. The domain $dom(\varphi)$ of a frame φ is the set of variables that φ exports.

An *evaluation context* is a context (a process with a hole) whose hole is not under a replication, a conditional, an input, or an output. A context $C[\_]$ closes A when $C[A]$ is closed.

The semantics of the applied pi calculus are defined by structural equivalence and internal reduction. *Structural equivalence* $\equiv$ is the smallest equivalence relation on extended processes that satisfies the following rules and that is closed under α-renaming of names and variables and under application of evaluation contexts.

$A \equiv A\|0$	PAR-0	$\nu n.0 \equiv 0$	NEW-0
$A\|(B\|C) \equiv (A\|B)\|C$	PAR-A	$\nu m.\nu n.A \equiv \nu n.\nu m.A$	NEW-C
$A\|B \equiv B\|A$	PAR-C	$\nu x.\{M/x\} \equiv 0$	ALIAS
$!P \equiv P\|!P$	REPL	$\{M/x\} \equiv \{N/x\}$, if $\Sigma \vdash M = N$	REWRITE
$\{M/x\}\|A \equiv A\{M/x\}$	SUBST		

Internal reduction $\rightarrow$ is the smallest relation on extended processes closed by structural equivalence and application of evaluation contexts such that:

$\overline{\mathsf{u}}\langle x\rangle.P \mid \mathsf{u}(x).Q \;\rightarrow\; P \mid Q$	COMM
$\texttt{if } M = M \texttt{ then } P \texttt{ else } Q \;\rightarrow\; P$	THEN
$\texttt{if } M = N \texttt{ then } P \texttt{ else } Q \;\rightarrow\; Q$ for all ground terms M, N s.t. $\Sigma \nvdash M = N$	ELSE

Observational equivalence is an important relation for the applied pi calculus. Intuitively, two processes are observationally equivalent if no evaluation context can distinguish them; evaluation contexts are often used to model attackers. We write $A \Downarrow a$ when A can send a message on a, i.e $A \rightarrow^* C[\bar{a}\langle M\rangle.P]$ for some evaluation context C that does not bind a.

Definition 1. *Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A \mathcal{R} B$ implies:*

1. *if $A \Downarrow a$, then $B \Downarrow a$;*
2. *if $A \rightarrow^* A'$, then $B \rightarrow^* B'$ and $A' \mathcal{R} B'$ for some B'.*
3. *$C[A] \mathcal{R} C[B]$ for all closing evaluation contexts $C[\_]$.*

Trace properties are also important in process calculi. *Correspondence* was introduced to capture these properties [8].

Definition 2 (Correspondence). *The closed process P satisfies the* correspondence*:*

$$P \Vdash \texttt{event}(f(x_1, \ldots, x_i)) \rightsquigarrow \texttt{event}(f'(y_1, \ldots, y_j))$$

means that if the event $f(x_1, \ldots, x_i)$ is executed, then event $f'(y_1, \ldots, y_j)$ must have been executed. The closed process P satisfies the injective correspondence*:*

$$P \Vdash \texttt{event}(f(x_1, \ldots, x_i)) \leftrightsquigarrow \texttt{event}(f'(y_1, \ldots, y_j))$$

means that for each event $f(x_1, \ldots, x_i)$ being executed, there is a unique event $f'(y_1, \ldots, y_j)$ which has been previously executed.

We refer the reader to [8] for the technical definition of correspondences.

2.2 Representing zero-knowledge proofs in applied pi

Zero-knowledge proofs become a widely used technique in constructing modern cryptographic protocols [18], including many credential systems. Loosely speaking, a zero-knowledge proof consists of a message or a sequence of messages that constitute a proof of a statement, which yields nothing but the validity of the statement. The applied pi calculus does not natively support the verification of security protocols involving zero-knowledge proofs, but Backes *et al.* have extended the tool ProVerif to enable modeling and analyzing non-interactive zero-knowledge proofs [5].

Let Σ_{base} be a base signature including logic and arithmetic operations as well as basic cryptographic primitives such as encryption, decryption, digital signature, etc., and E_{base} be an equational theory for Σ_{base}. For representing zero-knowledge, we need to extend the equational theory, based on an extended signature:

$$\Sigma_{ZK} = \Sigma_{base} \cup \{\mathtt{ZK}_{i,j}, \mathtt{Ver}_{i,j}, \mathtt{Public}_i, \mathtt{Formula}, \mathtt{true} \mid i, j \in \mathbb{N}\}$$

A non-interactive zero-knowledge proof is formalized as a term $\mathtt{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)$, where $\widetilde{M}$ denotes the term sequence $M_1, \ldots, M_i$ which represent the *private components* of the statement that are not revealed to the verifier and the adversary, and $\widetilde{N}$ denotes the term sequence $N_1, \ldots, N_j$ which represent the *public components* of the statement, and F constitutes a formula over these terms. In particular, we fix a distinguished set of variables $ZV = \{\alpha_1, \alpha_2, \ldots, \beta_1, \beta_2, \ldots\}$ which are only used to construct zero-knowledge formulas. Intuitively, α variables can be substituted by private components and β variables by public components. We call a term F an (i, j)*-formula* if it contains no names and $fv(F) \subseteq \{\alpha_1, \ldots, \alpha_i, \beta_1, \ldots, \beta_j\}$. $\mathtt{Public}_i, \mathtt{Formula}$ are operations for retrieving, respectively, the i-th public element and the formula from a proof, and $\mathtt{Ver}_{i,j}$ is the function which verifies a proof against a formula. We shall often omit the arities of $\mathtt{ZK}_{i,j}$ and $\mathtt{Ver}_{i,j}$ when they are clear from the context.

The equational theory E_{ZK} for representing zero-knowledge is the smallest theory satisfying all equations in E_{base} and the following equations defined over all terms $\widetilde{M}, \widetilde{N}, F$:

$$\begin{array}{lll}
\mathtt{Public}_l(\mathtt{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)) & = N_l, & 1 \leq l \leq j, \\
\mathtt{Formula}(\mathtt{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)) & = F, & \\
\mathtt{Ver}_{i,j}(F, \mathtt{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)) & = \mathtt{true} & \text{iff } E_{ZK} \vdash F\{\widetilde{M}/\widetilde{\alpha}\}\{\widetilde{N}/\widetilde{\beta}\} = \mathtt{true} \\
 & & \quad \text{and } F \text{ is an } (i, j)\text{-formula.}
\end{array}$$

Backes *et al.* also supply several techniques for dealing with infinite equational theories, so as to enforce the termination of the verification in ProVerif. We refer the reader to [18] for details.

3 A general description of credential systems in applied pi

In general, a credential system consists of two types of agents: *users* who wish to anonymously prove part of their personal information or use valuable services through credentials, and *organizations* who issue credentials to users and verify the validity of credentials shown by users. In *anonymous* credential systems, a user needs first to interact with an organization to establish a pseudonym before demanding a credential. The user is then known at the organization by the pseudonym, which is usually based on some information known by the organization about the user (e.g. an account in a bank). A real individual can have a pseudonym at each organization, or even multiple pseudonyms at one organization. However, organizations should not be able to link two different pseudonyms belonging to the same user. Credentials are issued by organizations to pseudonyms instead of real identities: when demanding a credential, the user interacts with the organization in the name of the pseudonym that he has established, and obtains a credential which can be shown by the user to another organization in the verifying procedure.

3.1 Modeling credential systems

We give a general framework of modeling credential systems in applied pi, by defining the overall structure of processes without concrete definitions. In later sections we shall see how a real credential system can be modeled following the structure. The model of a credential system in applied pi generally consists of two types of processes: the user processes and the organization processes.

When a user enters the system, she must first demand a pseudonym at some organization, and then use this pseudonym to demand and show credentials, hence a user process can be generally defined as

$$UP \stackrel{\text{def}}{=} \nu(\widetilde{\mathsf{u}}) \,.\, !\mathsf{ckey}(j, pk_j) \,.\, UN(j, pk_j) \,.\, (!UC(j, pk_j, nym_j) \mid !UV(j, pk_j, nym_j))$$

$\widetilde{\mathsf{u}}$ is a set of secret channels inside the user process which are basically used to transmit secret data like keys, randoms, and so on. The process $UN(j, pk_j)$ models the user's behavior of establishing a pseudonym at the organization O_j and the user must receive the correct public key pk_j properly (e.g., from a secret channel ckey shared between users and organizations). The process $UC(j, pk_j, nym_j)$ models the user's behavior of demanding a credential from the organization O_j, in name of the pseudonym nym_j that has been established in $UN(j, pk_j)$. In the end of the process, the generated credential must be recorded together with the ID of the issuing organization. The process $UV(j, pk_j, nym_j)$ models the user's behavior of showing a credential using the pseudonym nym_j. There are in general two manners of showing a credential:

$$\begin{aligned} UV(j, pk_j, nym_j) \stackrel{\text{def}}{=}\; & !\mathsf{u_i}(= j, cred_j) \,.\, UV^1(j, pk_j, cred_j) \\ & \mid !\mathsf{u_i}(l, cred_l) \,.\, UV^2(j, pk_j, nym_j, l, pk_l, cred_l), \end{aligned}$$

where $UV^1(j, pk_j, cred_j)$ models the behavior of showing a single credential $cred_j$ issued by the organization O_j and $UV^2(j, pk_j, nym_j, l, pk_l, cred_l)$ models the behavior of showing a credential $cred_l$ issued by the organization O_l, using the pseudonym nym_j (known at O_j. Note that in the first procedure, the credential can be essentially shown to any valid organization, while in the latter it can only be shown to O_j, i.e., the organization who knows the pseudonym nym_j. If the system guarantees unlinkability of pseudonyms, this does not break the anonymity.

Correspondingly, an organization process consists of generating a pseudonym, issuing a credential and verifying a credential:

$$OP \stackrel{\text{def}}{=} \nu(\widetilde{\mathsf{o}}) \,.\, !ON \,.\, (!OC(nym) \mid !OV^1(l, pk_l, cred) \mid !OV^2(l, pk_l, nym, cred)),$$

where the process ON models the organization's behavior of establishing a pseudonym nym, $OC(nym)$ models the behavior of issuing a credential to nym, $OV^1(l, pk_l, cred)$ models the behavior of verifying the credential $cred$ and $OV^2(l, pk_l, nym, cred)$ models the behavior of verifying the credential $cred$, plus the statement that its owner has the pseudonym nym. Note that nym in UV^2 must be the pseudonym established by the organization, but $cred$ in UV^1 and UV^2 can be an arbitrary credential issued by a valid organization (presumably the organization O_l). It is possible that some organizations only do the verification and never issue credentials, and in modeling a concrete system, one can safely remove the corresponding processes.

The whole credential system is then modeled as a set of user processes and organization processes running in parallel.

3.2 Events

As we shall see in Section 4, many security properties are defined using the notion of *correspondence* between events, which must be added at right places when we define processes. We summarize here a set of events which can be commonly defined in many credential systems and are sufficient for defining and verifying their security properties.

- **NymGenerated**(U, n): The user U executes this event when she establishes a pseudonym n with some organization.
- **NymApproved**(O, n): The organization O executes this event when he approves that the pseudonym n is correctly formed.
- **CredIssued**(O, c, n): The organization O executes this event after she issues the credential c to the pseudonym n.
- **UserShow**(U, c): A user executes this event when he starts a session of showing a credential c with a verifying organization.
- **CredVerified**(O, c, O'): Verifier O executes this event after the credential c has been shown to her and she is convinced that c has been issued by organization O'.
- **CredNymVerified**(O, n, c, O'): The verifying organization O executes this event after the credential c has been shown to her with the pseudonym n and she is convinced that c has been issued by O' to the user. Note that n is not the pseudonym to which c has been issued to, but the one that is known by the verifier O. In principle, the user must possess another pseudonym n' (known by O')and has used n' to get the credential c before she shows it, but this pseudonym is irrelvant in the verifying procedure. in short, n hides the user's identity at the verifying organization and n' protects her at the issuing organization.

We remark that events are not necessary for modeling protocols, but rather for specifying security properties based on traces, so only processes representing honest agents will execute these events — adversaries never execute events. When defining security properties using these events, we often omit some parameters (replaced by _) when they are irrelevant.

4 Security analysis of an anonymous credential system

In this section we apply the general modeling of the previous section to the anonymous credential system proposed by Camenisch and Lysyanskaya [10], and do a verification using ProVerif. The basic system consists of four protocols for, respectively, pseudonym generation, credential generation, showing a single credential and showing a credential w.r.t. a pseudonym. Due to the page limit, we only present here an abstract definition of the whole system. Detailed description of protocols and corresponding process specifications in applied pi can be found in Appendix A.

Definition 3 (Basic credential system). *The* basic credential system *of the CL system is a process in applied pi:*

$$BCS \stackrel{\text{def}}{=} \nu(\mathsf{ckey}).(UP_1 \mid \ldots \mid UP_n \mid OP_1 \mid \ldots \mid OP_m),$$

where ckey *is a secret channel for transferring organizations' public keys between honest agents,* $UP_i \stackrel{\text{def}}{=} \nu(\mathsf{cu}_i).!(\mathsf{ckey}(l, pk_l).UN_i(l, pk_l))$*, modeling each user agent, and*

$$OP_j \stackrel{\text{def}}{=} \nu(seed_j).\,\mathtt{let}\ pk_j = \mathtt{pkey}(seed_j),\ sk_j = \mathtt{skey}(seed_j)\ \mathtt{in}$$
$$!\overline{\mathsf{ckey}}(j, pk_j) \mid !\overline{\mathsf{c}}(j, pk_j) \mid !ON_j \mid !VP_j$$

modeling each organization agent, where c *is a public channel allowing agents including adversaries to communicate with each other.*

Definitions of the processes UN_i, ON_j, VP_j can be found in Appendix A.

The rest of the section is devoted to the formalization and verification of basic security properties: unforgeability of credentials and pseudonyms, safety of credentials and user privacy (pseudonym untraceability), which are supposed to be met by the CL basic system.

Let CS be a model of a credential system defined in applied pi, such as BCS in Definition 3. We write $\mathcal{U}(CS)$ for the set $\{U_1, \ldots, U_n\}$ where each U_i is a process representing an honest user agent, and $\mathcal{O}(CS)$ for the set $\{O_1, \ldots O_m\}$ where each O_j is a process representing an honest organization agent.

4.1 Unforgeability

Unforgeability of pseudonyms and credentials is the very basic security requirement of anonymous credential systems, which in principle prevents adversaries from forging fake credentials. Fake pseudonyms must be prevented too, since credentials are issued to pseudonyms, never to real identities.

Definition 4 (Pseudonym unforgeability). *A credential system CS respects* pseudonym unforgeability *if whenever an organization* $O' \in \mathcal{O}(CS)$ *issues a credential c to a pseudonym* n'*, she must have established this pseudonym with a user:*

$$CS \Vdash \mathtt{CredIssued}(O', c, n') \rightsquigarrow \mathtt{NymApproved}(O', n'), \tag{1}$$

and whenever an organization $O \in \mathcal{O}(CS)$ verifies a credential c that is shown to her w.r.t. a pseudonym n and is claimed to be issued by O', the verifier must have established the pseudonym with the user:

$$CS \Vdash \mathtt{CredNymVerified}(O, n, c, O') \rightsquigarrow \mathtt{NymApproved}(O, n), \tag{2}$$

Definition 5 (Credential unforgeability). *A credential system CS respects* credential unforgeability *if every successful showing (either single or with a pseudonym n at organization O) of a credential c, being claimed to be issued by an organization $O' \in \mathcal{O}(CS)$, implies that O' has previously issued c to some pseudonym n':*

$$CS \Vdash \mathtt{CredVerified}(\_, c, O') \rightsquigarrow \mathtt{CredIssued}(O', c, n'). \tag{3}$$

$$CS \Vdash \mathtt{CredNymVerified}(O, n, c, O') \rightsquigarrow \mathtt{CredIssued}(O', c, n'). \tag{4}$$

If a credential system respects both pseudonym unforgeability and credential unforgeability, we say that it is an *unforgeable credential system*.

The definition of credential unforgeability does not exclude the case where the adversary can forge a *valid* credential that has indeed been generated by an honest organization, but not to the adversary. We shall consider it as the safety of credentials.

Theorem 1. *The CL basic credential system (Definition 3) is an unforgeable credential system, i.e., it respects the unforgeability of both pseudonyms and credentials.*

Proof. We check the four correspondences (1), (2), (3) and (4) in ProVerif. □

4.2 Credential safety

Credential *safety* aims at preventing adversaries from *stealing* or using unauthorizedly a valid credential. In other words, no one other than the honest user, to whom a valid credential has been issued to, can successfully show the credential to a verifier. However, there is a subtle situation where safety can be confused with unforgeability: if an adversary can forge a credential which has been generated by an honest organization to an honest user, we shall consider it as an attack to safety instead of unforgeability. In fact, when we talk about credential safety, we actually mean safety of *unforgeable credentials*, or safety in *unforgeable credential systems*.

Definition 6 (Credential safety). *A credential c in an unforgeable credential system CS is* safe *if there is an injective correspondence between the event indicating that c (being issued by $O \in Org(CS)$) is successfully verified, and the event indicating that some user $U \in \mathcal{U}(CS)$, who must be the owner of the credential, starts to show c, i.e, for all pseudonym n',*

$$\begin{aligned} &CS \Vdash \mathtt{CredVerified}(\_, c, O') \rightsquigarrow \mathtt{CredIssued}(O', c, n') \\ &\quad \Rightarrow CS \Vdash \mathtt{CredVerified}(\_, c, O') \leftrightsquigarrow \mathtt{UserShow}(U, c) \\ &\qquad\qquad \wedge \mathtt{NymApproved}(O', n') \rightsquigarrow \mathtt{NymGenerated}(U, n'). \end{aligned} \tag{5}$$

If c is shown with respect to a pseudonym n at $O \in \mathcal{O}(CS)$, then the user must be the owner of n, i.e., there exists another pseudonym n',

$$
\begin{aligned}
CS \Vdash\ & \mathtt{CredNymVerified}(O, n, c, O') \rightsquigarrow \mathtt{CredIssued}(O', c, n') \\
\Rightarrow CS \Vdash\ & \mathtt{CredNymVerified}(O, n, c, O') \leftrightsquigarrow \mathtt{UserShow}(U, c) \\
& \wedge \mathtt{NymApproved}(O', n') \rightsquigarrow \mathtt{NymGenerated}(U, n') \\
& \wedge \mathtt{NymApproved}(O, n) \rightsquigarrow \mathtt{NymGenerated}(U, n)
\end{aligned}
\tag{6}
$$

If an unforgeable credential system respects credential safety, we say it is a *safe credential system*.

Unfortunately, the CL basic system is not safe if we assume the channels between users and organizations is insecure as in normal networking environment. There is a replay attack to safety, which works as follows: in the case of showing a single credential, the adversary can record A, B in Protocol 3 and all messages in the zero-knowledge proof, then sends them repeatedly to a verifier. The reused zero-knowledge proof simply passes. ProVerif actually shows that the injective correspondence in (5) fails. The same attack simply applies in the case of showing a credential w.r.t. a pseudonym, but the adversary can only show the credential to the organization whom the user has shown it to, since she cannot change the pseudonym that is used in the verification.

Note that an adversary can steal a pseudonym too and even use it to demand a new credential, but she cannot show it as in that case she cannot forge a proof for showing the credential. In fact, in credential systems, what we care about is what people can do with credentials, not what they can do with pseudonyms, so we do not define a property like pseudonym safety.

4.3 Pseudonym untraceability

Anonymous credential systems are designed essentially for providing user privacy. Credentials are issued to pseudonyms, so user privacy in credential systems indeed depends on what we can deduce based on pseudonyms. In particular, organizations should not be able to collectively distinguish pseudonyms that belong to different users. We call this property *pseudonym untraceability*, and in the applied pi calculus, this is formalized by the popular notion of *observational equivalence*.

Definition 7 (Pseudonym-untraceability). *A credential system CS respects* pseudonym untraceability *if for arbitrary users $U_i, U_j \in \mathcal{U}(CS)$ and a well formed public key pk_0,*

$$UN_i(\_, pk_0) \approx UN_j(\_, pk_0),$$

where UN_i (resp. UN_j), as defined in Definition 3, models the procedure of establishing a pseudonym by the user U_i (resp. U_j) and all her behavior involving the pseudonym in the system.

Theorem 2. *The basic credential system respects pseudonym untraceability, i.e. $UN_i(l, pk_l) \approx UN_j(l, pk_l)$.*

Proof. ProVerif supports proving observational equivalence of two processes which differ only in the choice of some terms. In our definition of pseudonym untraceability, we are actually proving: for all $U_i, U_j \in \mathcal{U}(BCS)$, $UN_i(0, pk_0) \approx UN_i(0, pk_0)[x_j/x_i]$. ProVerif shows that the above observational equivalence holds. $\square$

5 Related work

A security model for anonymous credential systems was proposed by Pashalidis and Mitchell [17]. It follows the idea of Bellare and Rogaway [6] based on complexity theoretic arguments, which potentially leads to information theoretic anonymity metrics. The model does not specify how the credential system achieves its goals but defines what the goals are. Some basic properties such as credential unforgeability, non-transferability, pseudonym unlinkability, and pseudonym owner protection are formally defined and the relationships between them are explored. Compared with our definitions, their definitions are based on a computational model while ours are based on the applied pi calculus and allow an automatic verification.

Abadi and Fournet introduced the applied pi calculus [3] as a language for reasoning about security protocols. The calculus inherits communication and concurrency for the pure pi calculus [16], and introduces functions and equations to reason about complex messages transmitted in security protocols. ProVerif [7] is an automatic cryptographic protocol verifier for the analysis of trace-based security properties and observational equivalence. It accepts applied pi processes as inputs and translates them into Horn clauses. Using this tool, Blanchet *et al.* verified a protocol for certified emails [1], a protocol for secure file sharing on untrusted storage [9], as well as the JFK protocol [2]. Luo *et al.* [15] analyzed an electronic cash protocol, and Kremer and Ryan [12] verified an electronic voting protocol. Backes *et al.* [5] introduced an implementation of zero-knowledge in equational theories acceptable by ProVerif and applied it to the analysis of a remote electronic voting protocol [4].

6 Conclusion

In this paper we have presented a general formalization of credential systems and some important security properties. We apply them to the concrete credential system proposed by Camenisch and Lysyanskaya and have verified that the basic system satisfies unforgeability for both pseudonyms and credentials, and pseudonym untraceability. We also reveal an attack to the system which allow adversaries to steal and unauthorizedly use a credential.

We argue that the model that we propose in the paper is faithful enough. However, as the original protocol itself is not written in a formal language, there is no way to formally prove that our model faithfully specify the original protocol. Nevertheless, if we assume that the specification is correct, then the soundness of ProtoVerif already guarantees the correctness of the verification output.

One novelty of the CL system, compared with other credential systems, is the implementation of non-transferable credentials which prevent users from lending their credentials. As part of the future work, we shall investigate how to formalize non-transferability, as well as other interesting, advanced properties like non-reshowability. We are also trying to transplant our model to other credential systems, in order to establish a scalable model of analyzing credential systems with the applied pi calculus. Another interesting work would be to focus on improving the efficiency for verifying complex systems using zero-knowledge proofs heavily. How to optimize equational theories to speed up termination is still a challenging problem.

References

1. Martín Abadi and Bruno Blanchet. Computer-Assisted Verification of a Protocol for Certified Email. In *Proceedings of the 10th International Symposium on Static Analysis*, volume 2694 of *Lecture Notes on Computer Science*, pages 316–335. Springer Verlag, 2003.
2. Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just fast keying in the pi calculus. In *Proceedings of the 13th European Symposium on Programming*, volume 2986 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 2004.
3. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, volume 36, pages 104–115. ACM, 2001.
4. Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium*, pages 195–209. IEEE Computer Society, 2008.
5. Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 202–215. IEEE Computer Society, 2008.
6. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Proceedings of the 13th Annual International Cryptology Conference*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.
7. Bruno Blanchet. Proverif: Cryptographic protocol verifier in the formal model. Available at `http://www.proverif.ens.fr/`.
8. Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 2008. To appear, Available at http://arxiv.org/abs/0802.3444.
9. Bruno Blanchet and Avik Chaudhuri. Automated formal analysis of a protocol for secure file sharing on untrusted storage. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 417–431. IEEE Computer Society, 2008.
10. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, 2001.
11. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
12. Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In *Proceedings of the 14th European Symposium on Programming*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200, 2005.
13. Xiangxi Li, Yu Zhang, and Yuxin Deng. ProVerif scripts for verifying a non-transferable anonymous credential system. Available at `http://basics.sjtu.edu.cn/~xiangxi/credentialsys.rar`.
14. Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.
15. Zhengqin Luo, Xiaojuan Cai, Jun Pang, and Yuxin Deng. Analyzing an electronic cash protocol using applied pi calculus. In *Proceedings of the 5th International Conference on Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2007.
16. Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
17. Andreas Pashalidis and Chris J. Mitchell. A security model for anonymous credential systems. In *Proceedings of the 19th International Workshop on Information Security*, pages 183–198. Kluwer, 2004.

18. Shafi Goldwasser Silvio Micali Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.
19. Kurt Seifried. The end of ssl and ssh? Available at `http://seifried.org/security/cryptography /20011108-end-of-ssl-ssh.html`.
20. Jean sbastien Coron, Marc Joye, David Naccache, Pascal Paillier, École Normale Supérieure, Gemplus Card International, and Gemplus Card International. New attacks on pkcs #1 v1.5 encryption. In *In Advances in Cryptology – Eurocrypt 2000*, pages 369–379. Springer Verlag, 2000.

A Description and modeling of the CL basic system

A.1 Setup of the CL system

The CL system uses the asymmetric cryptography (typically RSA) to implement anonymous pseudonyms and credentials. Each organization O_j will have a public key PK_j consisted of an RSA modulus n_j, and five elements of $QR_{n_j} : (a_j, b_j, d_j, g_j, h_j)$, the corresponding secret key that contains the factorization of n_j. Each user U_i has a master secret key x_i.

A pseudonym N_{ij} — a name for U_i being known at O_j — consists of a user-generated part N_1 and organization-generated part N_2. Every pseudonym N_{ij} will be tagged with a *validating tag* P_{ij}. A credential issued by O_j to a pseudonym N_{ij} is pair (e, c), where e is a sufficiently long prime chosen by O_j, and $c = P_{ij}d_j{}^{1/e} \pmod{n_j}$. Under the strong RSA assumption, such tuples cannot be existentially forged for correctly formed tags even by an adaptive attack, since no one can generate c from e without knowing the factorization of n_j.

Zero knowledge is applied in the system to protect users' privacy. A proof of possession of a credential is realized by a proof of knowledge of a correctly formed tag P_{ij} and a credential on it. This is done by publishing statistically secure commitments to both the validating tag and the credential, and proving relationships between these commitments. It can also include a proof that the underlying secret key is the same in both the committed validating tag (corresponding to the pseudonym formed with the issuing organization) and the validating tag with the verifying organization.

The base signature for analyzing the CL system consists basically of two sorts of functions: basic crytpographic primitives (e.g., `enc`, `dec` for encryption and decryption and `pkey`, `skey` for generating asymmetric key pairs) and cyclic group arithmetic operations (e.g., `add`, `mult`, `exp` and `inv` for group addition, multiplication, exponentiation and inverse operation). The equational theory for this signature contains standard equations for cryptography and RSA arithmetic. Detailed definition can be found in [13].

A.2 Basic credential system and its model in applied pi

The CL basic system consists of four protocols for, respectively, pseudonym generation, credential generation, showing a single credential and showing a credential w.r.t. a pseudonym. We briefly describe these protocols and give the user and organization processes corresponding their behavior in each protocol.

Protocol 1 (Pseudonym generation) User U_i follows the protocol below to establish a pseudonym at organization O_j:

1. U_i chooses values N_1, r_1, r_2, r_3, sets $C_1 = g_j^{r_1} h_j^{r_2}, C_2 = g_j^{x_i} h_j^{r_3}$ and sends N_1, C_1, C_2 to O_j. U_i proves that C_1 and C_2 are formed correctly in
$$PK\{(\alpha_1, \alpha_2, \alpha_3, \alpha_4) : C_1 = g_j^{\alpha_1} h_j^{\alpha_2} \wedge C_2 = g_j^{\alpha_3} h_j^{\alpha_4}\}.$$
2. O_j generates two randoms r, N_2 and sends them to U_i.
3. U_i computes $s_{ij} = r_1 + r$, sets the pseudonym $N_{ij} = (N_1, N_2)$ and the validating tag $P_{ij} = a_j^{x_i} b_j^{s_{ij}}$, then sends P_{ij} to O_j. U_i proves that P_{ij} is formed correctly in
$$PK\{(\alpha, \beta, \gamma, \delta, \varepsilon) : C_1 = g_j^{\alpha} h_j^{\beta} \wedge C_2 = g_j^{\gamma} h_j^{\delta} \wedge P_{ij} = a_j^{\gamma} b_j^{\varepsilon}\}$$

In this protocol, every generated pseudonym N_{ij} corresponds to a validating tag P_{ij}, and they are stored in pair by both U_i and O_j. Here P_{ij} is used to distinguish different pseudonyms, and the representation of P_{ij} with respect to g_j and h_j is an essential part when showing a credential.

The user process and the organization process for the protocol are defined as:

$$\begin{aligned}
UN_i(j, pk_j) \stackrel{\text{def}}{=} \; & \nu(N_1, r_1, r_2, r_3) \,. \\
& \texttt{let } C_1 = \texttt{exp}((g_j, h_j), (r_1, r_2)),\; C_2 = \texttt{exp}((g_j, h_j), (x_i, r_3)) \texttt{ in} \\
& \texttt{let } zp_1 = \mathsf{ZK}(r_1, r_2, x_i, r_3; C_1, C_2, g_j, h_j; F_1) \texttt{ in} \\
& \overline{\mathsf{co}_j}((N_1, C_1, C_2), zp_1) \,.\, \mathsf{cu}(r, N_2) \,. \\
& \texttt{let } N_{ij} = (N_1, N_2),\; s_{ij} = r_1 + r,\; P_{ij} = \texttt{exp}((a_j, b_j), (x_i, s_{ij})) \texttt{ in} \\
& \texttt{let } zp_2 = \mathsf{ZK}(r_1, r_2, x_i, r_3, s_{ij}; C_1, C_2, g_j, h_j, a_j, b_j, P_{ij}; F_2) \texttt{ in} \\
& \mathbf{NymGenerated}(U_i, N_{ij}) \,.\, \overline{\mathsf{co}_j}(P_{ij}, zp_2) \,. \\
& (!UC_i(j, N_{ij}, P_{ij}) \mid !(\mathsf{cu}_i(l, cred) \,.\, UV_i^2(j, l, cred, N_{ij}))), \\
ON_j \stackrel{\text{def}}{=} \; & \mathsf{co}_j(N_1, C_1, C_2, z_1) \,. \\
& \texttt{if Ver}(F_1, z_1) = \texttt{true then} \\
& \nu(r, N_2) \,.\, \overline{\mathsf{cu}}(\langle r, N_2 \rangle) \,.\, \mathsf{co}_j(P, z_2) \,. \\
& \texttt{if Ver}(F_2, z_2) = \texttt{true then} \\
& \texttt{let } N = (N_1, N_2) \texttt{ in} \\
& \mathbf{NymApproved}(O_j, N) \,.\, (!OC_j(N, P) \mid !OV_j(N, P))
\end{aligned}$$

where

$$\begin{aligned}
F_1 &\stackrel{\text{def}}{=} (\beta_1 = \texttt{exp}((\beta_3, \beta_4), (\alpha_1, \alpha_2)) \wedge \beta_2 = \texttt{exp}((\beta_3, \beta_4), (\alpha_3, \alpha_4))), \\
F_2 &\stackrel{\text{def}}{=} (\beta_1 = \texttt{exp}((\beta_3, \beta_4), (\alpha_1, \alpha_2)) \wedge \beta_2 = \texttt{exp}((\beta_3, \beta_4), (\alpha_3, \alpha_4)) \\
&\qquad \wedge \beta_7 = \texttt{exp}((\beta_5, \beta_6), (\alpha_3, \alpha_5))),
\end{aligned}$$

and co_j and cu are public channels, cu_i is a secret channel inside the process of U_i. $UC_i(j, N_{ij}, P_{ij})$ is the user process of demanding a credential from O_j, using the pseudonym

N_{ij} (together with the validating tag P_{ij}). $UV_i^2(j,l,cred,N_{ij})$ is the user process of showing, to organization O_j, the credential *cred* issued by organization O_l, using the pseudonym N_{ij}. $OC_j(N,P)$ and $OV_j(N,P)$ represent, respectively, the organization processes of issuing a credential to the pseudonym N and of verifying a credential sent by a user using the pseudonym N. Note that the credential issue and verification can be done separately at the organization side (by different processes in parallel), but they must be done sequentially in the user process.

The events **NymGenerated** and **NymApproved** are executed in this protocol: **NymGenerated** is executed by the user right after he receives the organization part of the pseudonym, and **NymApproved** is executed by the organization at the end of the protocol, when he receives the validating tag and verifies the validity of its form.

Note that the user is assumed to communicate with the organization via an *anonymous* channel, which is modeled using a global public channel c, and we rely on the scheduler to determine the right destination of messages transmitted on the channel. In particular, there is no successful trace where the response of an organization is sent to a wrong user, as in that case the organization will fail in checking the the second zero-knowledge proof.

Protocol 2 (Credential generation) User U_i follows the protocol below to demand a credential from organization O_j:

1. U_i sends (N_{ij}, P_{ij}) to O_j and proves the ownership in
$$PK\{(\alpha,\beta) : P_{ij} = a_j^\alpha b_j^\beta\}.$$
2. O_j checks that (N_{ij}, P_{ij}) is in its database, chooses a random prime e, computes $c = (P_{ij}d_j)^{1/e} \mod n_j$, sends c and e to U_i and stores (c, e) in its record for N_{ij}.
3. U_i checks if $c^e = P_{ij}d_j \mod n_j$; if so, she stores (c, e) in its record with organization O_j. The tuple (c, e) is called a *credential record*.

The cryptographic assumption ensures that an adversary who does not know the factorization of n_j should not be able to generate c from e.

The user process and the organization process of credential generation are:

$$\begin{aligned}
UC_i(j, N_{ij}, P_{ij}) \overset{\text{def}}{=}\ & \mathtt{let}\ zp_3 = \mathsf{ZK}(x_i, s_{ij}; P_{ij}, a_j, b_j; F_3)\ \mathtt{in} \\
& \overline{\mathsf{co}_j}(N_{ij}, P_{ij}, zp_3)\,.\,\mathsf{cu}(c_{ij}, e_{ij})\,. \\
& \mathtt{if}\ \mathtt{exp}(c_{ij}, e_{ij}) = \mathtt{mult}(P_{i,j}, d_j)\ \mathtt{then} \\
& \mathtt{let}\ cred_{ij} = (c_{ij}, e_{ij})\ \mathtt{in} \\
& !\overline{\mathsf{cu}_i}(j, cred_{ij}) \mid !UV_i^1(j, cred_{ij})
\end{aligned}$$

$$\begin{aligned}
OC_j(N,P) \overset{\text{def}}{=}\ & \mathsf{co}_j(N', P', z_3)\,. \\
& \mathtt{if}\ \mathtt{Ver}(F_3, z_3) = \mathtt{true}\ \mathtt{then} \\
& \nu(e)\,.\,\mathtt{let}\ c = \mathtt{exp}(\mathtt{mult}(P, d_j), \mathtt{inv}(e)),\ cred = (c, e)\ \mathtt{in} \\
& \qquad \mathbf{CredIssued}(O_i, cred, N)\,.\,\overline{\mathsf{cu}}(c, e),
\end{aligned}$$

where
$$F_3 \overset{\text{def}}{=} \beta_1 = \mathtt{exp}((\beta_2, \beta_3), (\alpha_1, \alpha_2)).$$

$UV_i^1(j, cred_{ij})$ is the user processes of showing the credential $cred_{ij}$ to an arbitrary organization (verifier).

When U_i receives a credential, she broadcasts it via the internal secret channel $\mathtt{cu}_i$ to all other sub-processes; when U_i wants to show a credential with respect to a pseudonym N_{ij}, she invokes the procedure UV_i^2 by sending the credential to the procedure via $\mathtt{cu}_i$.

The event **CredIssued** is executed by the organization in this protocol after the credential is generated.

Protocol 3 (Showing a single credential) User U_i follows the protocol below to show a credential, issued by O_j, to a verifier V (without revealing the combined pseudonym):

1. U_i chooses r'_1, r'_2, computes $A = c_{ij} h_j^{r'_1}$, $B = h_j^{r'_1} g_j^{r'_2}$, and sends A, B with the credential to V.
2. U proves the validity of the credential in

$$PK\{(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7) : \\ d_j = A^{\alpha_1}(\frac{1}{a_j})^{\alpha_2}(\frac{1}{b_j})^{\alpha_3}(\frac{1}{h_j})^{\alpha_4} \wedge B = h_j^{\alpha_5} g_j^{\alpha_6} \wedge 1 = B^{\alpha_1}(\frac{1}{h_j})^{\alpha_4}(\frac{1}{g_j})^{\alpha_7}\}$$

Those who can successfully show a single credential are assumed to know the representation of P_{ij} with respect to g_j, h_j as well as the credential pair (c_{ij}, e_{ij}), so the protocol should offer sufficiently security even when the transmission of a credential is unsafe.

The two processes engaged in this protocol are:

$$\begin{aligned} UV_i^1(j, cred_{ij}) \stackrel{\text{def}}{=}\ & \nu(r'_1, r'_2)\,. \\ & \mathtt{let}\ A = \mathtt{exp}((c_{ij}, h_j), (1, r'_1)),\ B = \mathtt{exp}((h_j, g_j), (r'_1, r'_2))\ \mathtt{in} \\ & \mathtt{let}\ zp_4 = \mathtt{ZK}(e_{ij}, x_i, s_{ij}, \mathtt{mult}(r'_1, e_{ij}), r'_1, r'_2, \mathtt{mult}(r'_2, e_{ij}); \\ & \qquad\qquad A, B, a_j, b_j, d_j, g_j, h_j; F_4)\ \mathtt{in} \\ & \mathbf{UserShow}(U_i, cred_{ij})\,.\,\overline{\mathtt{cv}}(j, c_{ij}, e_{ij}, A, B, zp_4), \end{aligned}$$

$$\begin{aligned} VP_j \stackrel{\text{def}}{=}\ & \mathtt{cv}(= j, c_{ij}, e_{ij}, A, B, z_4)\,. \\ & \mathtt{if}\ \mathtt{Ver}(F_4, z_4) = \mathtt{true}\ \mathtt{then}\ \mathbf{CredVerified}(\_, cred, O_j), \end{aligned}$$

where

$$\begin{aligned} F_4 \stackrel{\text{def}}{=}\ & \beta_5 = \mathtt{exp}((\beta_1, \mathtt{inv}(\beta_3), \mathtt{inv}(\beta_4), \mathtt{inv}(\beta_7)), (\alpha_1, \alpha_2, \alpha_3, \alpha_4)) \\ & \wedge \beta_2 = \mathtt{exp}((\beta_7, \beta_6), (\alpha_5, \alpha_6)) \wedge 1 = \mathtt{exp}((\beta_2, \mathtt{inv}(\beta_7), \mathtt{inv}(\beta_6)), (\alpha_1, \alpha_4, \alpha_7)) \end{aligned}$$

The events **UserShow** and **CredVerified** are executed in this protocol: **UserShow** is executed by the user right before he starts to show a credential and **CredVerified** is executed by the verifier after verifying the validity of the credential (the first parameter is omitted since in this protocol the identity of the verifier is irrelevant). In the process model, we explicitly let the user transmit the credential to the verifier, which is not included in the original protocol. This is only for the verifier process to be able to execute the event **CredVerified**. It is no harm of sending the credential over a public channel since any valid credential has been sent over a public channel when it is first generated.

Protocol 4 (Showing a credential w.r.t. a pseudonym) User U_i follows the protocol below to show a credential issued by organization O_l, to another organization O_j, using a pseudonym N_{ij} that she has established with O_j:

1. U_i chooses r'_1, r'_2, computes $A = c_{il}\, h_l^{r'_1}$ and $B = h_l^{r'_1} g_l^{r'_2}$, and sends N_{ij}, A, B to O_j.
2. U proves the validity of the credential and the ownership of N_{ij} in

$$PK\{(\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5,\alpha_6,\alpha_7,\alpha_8) : \\ d_l = A^{\alpha_1}(\frac{1}{a_l})^{\alpha_2}(\frac{1}{b_l})^{\alpha_3}(\frac{1}{h_l})^{\alpha_4} \wedge B = h_l^{\alpha_5} g_l^{\alpha_6} \wedge 1 = B^{\alpha_1}(\frac{1}{h_l})^{\alpha_4}(\frac{1}{g_l})^{\alpha_7} \wedge P_{ij} = a_j^{\alpha_2} b_j^{\alpha_8}\}$$

The above proof has a fourth equation which proves that the same master secret key that is used in constructing the credential c_{il} (issued by O_l), is also used in P_{ij}, the attached validating tag of the pseudonym N_{ij} which is established with O_j.

The two processes engaged in this protocol are:

$$\begin{aligned} UV_i^2(j, l, cred, N_{ij}) \stackrel{\text{def}}{=}\ & \nu(r'_1, r'_2)\,. \\ & \mathtt{let}\ A = \mathtt{exp}((c_{ij}, h_j), (1, r'_1)),\ B = \mathtt{exp}((h_j, g_j), (r'_1, r'_2))\ \mathtt{in} \\ & \mathtt{let}\ zp_5 = \mathsf{ZK}(e_{il}, x_i, s_{il}, \mathtt{mult}(r'_1, e_{il}), r'_1, r'_2, \mathtt{mult}(r'_2, e_{il}), s_{ij}; \\ & \qquad\qquad A, B, a_l, b_l, d_l, g_l, h_l, P_{ij}, a_j, b_j; F_5)\ \mathtt{in} \\ & \mathbf{UserShow}(U_i, cred)\,.\,\overline{\mathsf{co}_j}(k, cred, A, B, N_{ij}, zp_5) \end{aligned}$$

$$\begin{aligned} OV_j(N, P) \stackrel{\text{def}}{=}\ & \mathsf{cv}(l, cred, A, B, = N, z_5)\,.\,\mathsf{ckey}(= l, pk_l)\,. \\ & \mathtt{if}\ \mathtt{Ver}(F_5, z_5) = \mathtt{true}\ \mathtt{then}\ \mathbf{CredNymVerified}(O_j, N, cred, O_l) \end{aligned}$$

where

$$\begin{aligned} F_4 \stackrel{\text{def}}{=}\ & \beta_5 = \mathtt{exp}((\beta_1, \mathtt{inv}(\beta_3), \mathtt{inv}(\beta_4), \mathtt{inv}(\beta_7)), (\alpha_1, \alpha_2, \alpha_3, \alpha_4)) \\ & \wedge \beta_2 = \mathtt{exp}((\beta_7, \beta_6), (\alpha_5, \alpha_6)) \wedge 1 = \mathtt{exp}((\beta_2, \mathtt{inv}(\beta_7), \mathtt{inv}(\beta_6)), (\alpha_1, \alpha_4, \alpha_7)) \\ & \wedge \beta_8 = \mathtt{exp}((\beta_9, \beta_{10}), (\alpha_2, \alpha_8)) \end{aligned}$$

The two processes are similar as those for Protocol 3, except that the user needs to send a pseudonym to the verifier and it involves in the zero-knowledge proof.

Similar as **CredVerified**, the event **CredNymVerified** is executed in this protocol by the verifier after the verification, but it contains more information than **CredVerified**.