Formal Analysis of a TTP-Free Blacklistable Anonymous Credentials System (Full Version)

Weijin Wang[1,2], Yu Qin[1], Jingbin Liu[1,2], and Dengguo Feng[1,3]

[1]TCA, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]SKLCS, Institute of Software, Chinese Academy of Sciences, Beijing, China
{wangweijin,Qin_Yu,liujingbin,feng}@tca.iscas.ac.cn

Abstract. This paper firstly introduces a novel security definition for BLAC-like schemes (BLAC represents TTP-free BLacklist-able Anonymous Credentials) in the symbolic model using applied pi calculus, which is suitable for automated reasoning via a certain formal analysis tool. We model the definitions of some common security properties: authenticity, non-framebility, mis-authentication resistance and privacy (anonymity and unlinkability). Then the case study of these security definitions is demonstrated by modelling and analyzing BLACR (BLAC with Reputation) system. We verify these security properties by Blanchet's ProVerif and a ZKP (Zero-Knowledge Proof) compiler developed by Backes *et al.*. In particular, we model and analyze the express-lane authentication in BLACR system. The analysis discovers a known attack that can be carried out by any potential user. This attack allows a user escaping from being revoked as he wishes. We provide a revised variant that can be proved successfully by ProVerif as well, which also indicates that the fix provided by ExBLACR (Extending BLACR) is incorrect.

Keywords: Formal analysis, Anonymous Credential, ProVerif, BLACR

1 Introduction

Anonymous credentials allow users to obtain credentials on their identities and prove possession of these credentials anonymously. There are three parties in the anonymous credentials system: *users* obtain credentials from *issuers* (or GM, indicating Group Manager). They can then present these credentials to *verifiers* (or SP, indicating Service Provider) in an anonymous manner. The verifiers can check the validity of users' anonymous credentials but cannot identify them.

Practical solutions for anonymous credentials have been proposed, such as IBM's identity mixer [25] and TCG (Trusted Computing Group)'s DAA (Direct Anonymous Attestation) protocol [20, 21] based on CL-signatures [26, 27], Microsoft's U-Prove [32] based on Brands' signatures [19], or Nymble system [29]. To avoid misbehavior, most of schemes introduce a TTP (Trust Third Party) to revoke misbehaved users. However, having a TTP capable of deanonymizing or linking users' access may be dangerous. Recognizing this, elimination of such

TTP while still supporting revocation is desired. In this spirit, many schemes had been proposed, such as EPID [22], BLAC [37], BLACR [9], ExBLACR [39], PEREA [7], PERM [8], PE(AR)^2 [42], FARB [40]. In these TTP-free schemes, SP can punish users without the assistance of TTP, and users must convince SP that they satisfy the predetermined authentication policy in a zero-knowledge way during authentication.

All these schemes are claimed to be provable secure except for U-Prove. However, computational security definitions of these schemes are very complex, thus making the proof of security error-prone. For example, Camenisch *et al.* [24] pointed out that the known security models of DAA are non-comprehensive and even insecure recently, and gave a security model under universally composable framework. BLACR system is also reported that a feasible attack exists [39]. Recognizing this, we tend to prove these complex schemes in another perspective, namely formal methods, which are widely used to verify cryptographic protocols. We think formal analysis can help us to find the logical errors of protocols and become a complement of the computational security proof.

Fortunately, formal analysis has shown its power to prove the complex security definitions although the formal analysis of anonymous credentials is relatively limited (almost for DAA). Arapinis *et al.* [5] presented a framework for analyzing the unlinkability and anonymity in the applied pi calculus. Arapinis *et al.* [3, 4] make use of this framework to analyze the privacy of composing protocols using ProVerif [17]. Smyth *et al.* [35, 36] introduced a definition of privacy for DAA schemes that was suited to automated reasoning by ProVerif. They discovered a vulnerability in the RSA-based DAA scheme and fixed it to meet their definition of privacy. Xi *et al.* [41] utilized ProVerif to analyze the DAA scheme in TPM 2.0. They put forward the definition of forward anonymity for the DAA scheme in symbolic model. To deal with the complex zero-knowledge proof within equational theory, Backes *et al.* [12] presented an abstraction of zero-knowledge proof that is formalized by the applied pi calculus and developed a compiler to encode this abstraction into a rewriting system that is suited to ProVerif. They also performed mechanized analysis of DAA using their approach and found a novel attack.

Contributions. In this paper, we present a novel definition for some common security properties of BLAC-like schemes via applied pi calculus. Specifically, we formalize authenticity, non-frameability and mis-authentication resistance as correspondence properties and privacy as equivalence properties using applied pi calculus (Section 3).

For a case study, we analyze BLACR system (Section 4). BLACR is a TTP-free blacklistable anonymous credentials system that allows SPs to revoke the misbehaved users directly according to their reputation scores (Section 4.2). We model its sub-protocols by applied pi processes and defined some main processes to analyze those security properties defined previously. Our analysis result shows that the BLACR holds these security properties in the normal-lane form (Section 4.4). Specially, we also model and analyze the express-lane authentication of BLACR (Section 4.5). This analysis shows an anticipative actions if a user

always does not trigger the revocation conditions and reports a known vulnerability when a user have potential to get revoked. This attack allows a user to escape from being revoked as he wishes after he owns a express-lane token, which disables the security policy of BLACR. Then we provide a revised variant that can be proved by ProVerif. The revision also shows that the fix provided by ExBLACR is incorrect.

2 Overview of ProVerif Calculus and ZKP compiler

We adopt the process calculus of ProVerif [13, 1, 14, 16, 28], which is inspired by the applied pi calculus [2, 33]. In the context of no ambiguity, we sometimes call it applied pi calculus instead of ProVerif calculus. We also utilize Backes' ZKP compiler to encode zero-knowledge proofs [12].

2.1 Overview of ProVerif Calculus

Figure 1 summarizes the syntax of the calculus. It assumes an infinite set of names, an infinite set of variables, and a signature Σ consisting of a finite set of function symbols (constructors and destructors). Constructors f are used to build terms while destructors g manipulate terms in processes. The signature Σ is equipped with a finite set of equations whose form is $M = N$.

A destructor g is defined by a finite set $\mathrm{def}(g)$ of rewriting rules $g(N_1, ..., N_n) \rightarrow N$ that processes can apply, where $N_1, ..., N_n, N$ are terms containing only constructors and variables. Function h represents either a constructor or destructor.

$M, N ::=$	terms
x, y, z	variable
a, b, c, k, s	name
$f(M_1, ..., M_n)$	constructor application
$D ::=$	term evaluations
M	term
$h(D_1, ..., D_n)$	function evaluation
$P, Q, R ::=$	processes
0	nil
$P \mid Q$	parallel composition
$!P$	replication
$va.P$	restriction
$M(x).P$	input
$\overline{M}\langle N \rangle.P$	output
$let\ x = D\ in\ P\ else\ Q$	term evaluation
$if\ M = N\ then\ P\ else\ Q$	conditional

Fig. 1. Syntax for ProVerif

Process P, Q are defined as follows. The null process 0 does nothing; $P|Q$ is the parallel composition of processes P and Q; and the replication $!P$ is the infinite composition of P in parallel; the restriction $va.P$ creates a new name a and binds a inside P; the process $M(x).P$ inputs a message on channel M and then behaves as P with the received message bound to x; the process $\overline{M}\langle N\rangle.P$ outputs the message N on channel M and then runs P. The process $let\ x = D\ in\ P\ else\ Q$ tries to evaluate D, if it succeeds, then x is bound to the result and P is executed, otherwise Q is executed. The conditional $if\ M = N\ then\ P\ else\ Q$ is in fact a particular case of destructor application, which is equivalent to $let\ x = eq(M, N)\ in\ P\ else\ Q$, where the destructor eq is defined by $eq(x, x) \to x$.

Figure 2 defines the operational semantics of the calculus by reduction relation ($\to_\Sigma$). Auxiliary rules define term evaluation ($\Downarrow_\Sigma$) and the structural equivalence ($\equiv$). We omit the operational semantic of process `conditional` since it is just a special case of term evaluation. Especially, $P\{M/x\}$ is a syntactic sugar of $let\ x = M\ in\ P$, where $\{M/x\}$ is a substitution that replaces x with M.

$$
\begin{array}{ll}
M \Downarrow M & \\
\text{eval } h(D_1, ..., D_n) \Downarrow \sigma N & \\
\quad \text{if } h(N_1, ..., N_n) \to N \in \text{def}(h), & \\
\quad \text{and } \sigma \text{ is such that for all } i, D_i \Downarrow M_i \text{ and } \Sigma \vdash M_i = \sigma N_i & \\
P|0 \equiv P & P \equiv P \\
P|Q \equiv Q|P & Q \equiv P \Rightarrow P \equiv Q \\
(P|Q)|R \equiv P|(Q|R) & P \equiv Q, Q \equiv R \Rightarrow P \equiv R \\
va.vb.P \equiv vb.va.P & P \equiv Q \Rightarrow P|R \equiv Q|R \\
va.(P|Q) \equiv P|va.Q & P \equiv Q \Rightarrow va.P \equiv va.Q \\
\quad \text{if } a \notin fn(P) & \\
\overline{N}\langle M\rangle.Q|\overline{N'}(x).P \to Q|P\{M/x\} & \\
\quad \text{if } \Sigma \vdash N = N' & \text{(Red I/O)} \\
let\ x = D\ in\ P\ else\ Q\ \to P\{M/x\} & \\
\quad \text{if } D \Downarrow M & \text{(Red Fun 1)} \\
let\ x = D\ in\ P\ else\ Q \to Q & \\
\quad \text{if there is no } M \text{ such that } D \Downarrow M & \text{(Red Fun 2)} \\
!P \to P|!P & \text{(Red Repl)} \\
P \to Q \Rightarrow P|R \to Q|R & \text{(Red Par)} \\
P \to Q \Rightarrow va.P \to va.Q & \text{(Red Res)} \\
P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q' & \text{(Red } \equiv\text{)}
\end{array}
$$

Fig. 2. Semantics for ProVerif

Correspondences. ProVerif can verify correspondences in security protocols. Correspondences are used to capture relationships between events that can be expressed in the form "if some event has been executed then some other events have been previously executed". In particular, correspondences can be used to

formalize authentication. To specify correspondences, the syntax in Figure 1 needs to provide an additional instruction $\texttt{event}(M).P$ for executing events. This additional instruction executes the event $\texttt{event}(M)$, then runs P. Besides, a new rule $\texttt{event}(M).P \to P$ is appended in the semantics. The details can be found in [15].

Observational equivalence. Some security properties, such as privacy-related properties, can be defined by the notion of observational equivalence. Roughly speaking, processes P and Q are said to be observationally equivalent if they can output on the same channel for all contexts they are placed inside. The formal definition of observational equivalence [16] is as follows.

Definition 1. *The process P emits on M ($P \downarrow_M$) if and only if $P \equiv C[\overline{M'}\langle N\rangle.R]$ for some evaluation context C that does not bind fn(M) and $\Sigma \vdash M = M'$.*

Observational equivalence $\sim$ *is the largest symmetric relation $\mathcal{R}$ on closed processes such that $P\ \mathcal{R}\ Q$ implies:*

1. *if $P \downarrow_M$ then $Q \downarrow_M$;*
2. *if $P \to P'$ then $Q \to Q'$ and $P'\ \mathcal{R}\ Q'$ for some Q';*
3. *$C[P]\ \mathcal{R}\ C[Q]$ for all evaluation contexts C.*

ProVerif cannot verify observational equivalence directly, so it introduces a new calculus *biprocess* to establish a sufficient condition for observational equivalence. A biprocess is a pair of processes that have the same structure and differ only by the terms and term evaluations that they contain. The syntax of biprocess is a simple extension of the Figure 1, with additional cases such that $\texttt{diff}[M, M']$ is a term and $\texttt{diff}[D, D']$ is a term evaluation. The semantics of biprocess is the same as Figure 2, except for rules (Red I/O), (Red Fun 1), (Red Fun 2) being replaced by Figure 3. Given a biprocess P, $\texttt{fst}(P)$ denotes a process replacing all occurrences of $\texttt{diff}[M, M']$ with M and $\texttt{diff}[D, D']$ with D in P, and similarly for $\texttt{snd}(P)$. Then observational equivalence can be defined as a property of biprocesses.

$$\overline{N}\langle M\rangle.Q|N'(x).P \to Q|P\{M/x\}$$
if $\Sigma \vdash \mathrm{fst}(N) = \mathrm{fst}(N')$ and $\Sigma \vdash \mathrm{snd}(N) = \mathrm{snd}(N')$ (Red I/O)

$$\textit{let } x = D \textit{ in } P \textit{ else } Q \to P\{\mathrm{diff}[M_1, M_2]/x\}$$
if $\mathrm{fst}(D) \Downarrow M_1$ and $\mathrm{snd}(D) \Downarrow M_2$ (Red Fun 1)

$$\textit{let } x = D \textit{ in } P \textit{ else } Q \to Q$$
if there is no M_1 such that $\mathrm{fst}(D) \Downarrow M_1$ *and* there is no M_2 such that $\mathrm{snd}(D) \Downarrow M_2$ (Red Fun 2)

Fig. 3. Generalized rules for biprocesses

Definition 2. *Let P be a closed biprocess. We say that P satisfies* observational equivalence *when* $\mathrm{fst}(P) \sim \mathrm{snd}(P)$.

2.2 Overview of ZKP Compiler

Figure 4 shows an equational theory for abstractly reasoning about non-interactive zero-knowledge proofs. The ZKP compiler can encode this equational theory into a convergent rewriting system that ProVerify can apply. A zero-knowledge proof has the form $\mathrm{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)$ ($\mathrm{ZK}(\widetilde{M}; \widetilde{N}; F)$ for short), where $\widetilde{M}$,$\widetilde{N}$ denote sequences $M_1, ..., M_i$,$N_1, ..., N_j$ and F denotes a formula over those terms. The terms $\widetilde{M}$ is the statement's *private component* while $\widetilde{N}$ is the statement's *public component* that will be revealed to the verifier and the adversary. The formula F is an (i, j)-formula that is defined as follow.

Definition 3 (**(i, j)-formulas**). *We call a term an* (i, j)-formula *if the term contains neither names nor variables, and if for every* α_k *and* β_l *occurring therein, we have* $k \in [1, i]$ *and* $l \in [1, j]$.

For example, the term $\mathrm{ZK}(x; g, y = g^x; \beta_2 = \beta_1^{\alpha_1})$ denotes a ZKP protocol that proves the knowledge of x such that $y = g^x$ holds. The public components y are revealed by the destructor $\mathtt{Public}_p$ and the formula $\beta_2 = \beta_1^{\alpha_1}$ is revealed by destructor `Formula`. Since there is no destructor associated to the private component, the secret of x is kept.

To verify the statement $\mathrm{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)$ with respect to a formula F, it defines a destructor function $\mathrm{Ver}_{i,j}$ of arity 2. This function is true when F is an (i, j)-formula and the formula is valid by substituting all α_k and β_l with the corresponding values M_k and N_l. This rule guarantees in the abstract model the *soundness* and *correctness* of zero-knowledge protocols.

$\Sigma_{\mathrm{ZK}} = \{\mathrm{ZK}_{i,j}, \mathrm{Ver}_{i,j}, \mathrm{Public}_i, \mathrm{Formula}, \alpha_i, \beta_j, \mathrm{true} | i, j \in N\}$,
where $ZK_{i,j}$ of arity $i + j + 1$, $\mathrm{Ver}_{i,j}$ *of* arity 2, Public_i and Formula of arity 1, α_i, β_j and true of arity 0.

E_{ZK} is the corresponding equational theory defined as follows :

$\mathrm{Public}_p(\mathrm{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)) = N_p$ with $p \in [1, j]$

$\mathrm{Formula}(\mathrm{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)) = F$

$\mathrm{Ver}_{i,j}(F, \mathrm{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)) = \mathrm{true}$ iff

1) $E_{ZK} \bot F\{\widetilde{M}/\widetilde{\alpha}\}\{\widetilde{N}/\widetilde{\beta}\} = \mathrm{true}$
2) F is an (i, j)-formula

Fig. 4. Equational theory for zero-knowledge

ProVerif cannot deal with this equational theory for zero-knowledge proofs because the signature Σ_{ZK}, and consequently the number of equations, is infinite (every zero-knowledge is different and needs a relevant signature). Backes *et al.* develop a compiler to encode zero-knowledge proof description into ProVerif-accepted specification, which can be found in [11].

3 Syntax and Security Definition

3.1 Syntax

Roughly speaking, a TTP-free blacklistable anonymous credentials system contains the following algorithms:

Initialization This algorithm initializes the system parameters. The issuer constructs a signing key pair (pk_I, sk_I). If SP is not the issuer, then SP will construct its own key pair (pk_V, sk_V). Especially, the implementation-specific parameters will be defined, such as initializing the blacklist.

Registration This algorithm is registration phase between the issuer and a legitimate user to enroll the user as a member in the group of registered users. Upon successful completion of this phase, The user obtains a credential signature *cre* on his secret value x. Usually, the GM issues the credential in a *blind* way.

Authentication The user will generate a zero-knowledge proof to convince an SP that he has the right to obtain service. First, the user in possession of x proves that he holds a valid credential *cre*. Then the user convinces that he satisfies the authentication policy of SP. Note that a *protocol transcript* τ (usually is a ticket) must be seen by the SP to guarantee freshness and to blacklist the authenticating user if necessary.

Verification SP will check the validity of the received zero-knowledge proofs. If failed, the user will be blocked to access.

List Management SP can manipulate the list with the transcript τ according to a specific authentication policy. In a reputation-based policy, the SP scores the user's action of the session with a transcript τ and executes the operation $add(L, (\tau, s))$ to add the score s to the current blacklist L.

3.2 Security Definition

In this section, we present the definitions of security properties in the symbolic model using applied pi calculus. We are involved in verifying some common properties of TTP-free blacklistable anonymous credentials system, namely *authenticity*, *non-frameability*, *mis-authentication resistance*, and *privacy*. Prior to this, appropriate assumptions and notations will be demonstrated.

Assumptions and Notations In this paper, we denote registration process as `Register` (for users) and `Issue` (for the issuer), and authentication process as `Authenticate`. Verification and list management processes can be combined together since they are all handled by SP, which is denoted as `Verify`. Initialization process will be encoded into the main process.

In process `Register`, event *registered*[1] will be executed after the user successfully registers with the issuer and obtains a valid credential, otherwise, event *unregistered* will be executed. In process `Authenticate`, event *startAuth* represents

[1] Actually, the event *e* always take some parameters as $e(M_1, M_2, ..., M_n)$. We omit the parameters here for convenient description and readability.

a new authentication activated by the user. Event *acceptAuth* will be executed when the verification of zero-knowledge proofs succeeds in process `Verify`, and conversely, event *revoke* will be executed, which means that the verification fails and the user has been revoked.

We assume that the adversary controls the execution of an arbitrary number of users in an arbitrary fashion except for learning their secret, as show in the following process:

$$\begin{aligned}&\texttt{ControlUsers} = !pub(id).\\ &\quad !vp.(\texttt{Register} \mid (p(cre).!(\texttt{Authenticate} \mid \texttt{Judge}))).\end{aligned}$$

The adversary can choose any user (id) to run the processes `Register` and `Authenticate`. The restricted channel name p is used for delivering the credential of the user between registration and authentication.

Process `Judge` models the judgment of a user's state lying in the authentication policy (for example, his current reputation score). We record two events in process `Judge`: event *satisfyPolicy* for a satisfied judgment; and event *notSatisfy* for a failure.

Authenticity In a system with *authenticity*, an SP is assured to accept authentication only from users who satisfy the authentication policy. This definition can be parsed as the following statements:

1. SP accepts authentication from users who satisfy the authentication policy.
2. SP would never accept authentication from users who violate the authentication policy.

Build on this understanding, we formalize authenticity (Definition 4) as two correspondence properties using the events recorded in the processes.

Definition 4 (Authenticity). *Given processes* ⟨`Register`,`Issue`,`Authenticate`, `Verify`,`Judge`⟩*, we say authenticity is satisfied if the following correspondences are held:*

$$\begin{aligned}&\textit{event:}\mathit{acceptAuth} \rightsquigarrow \mathit{startAuth} \;\&\; \mathit{satisfyPolicy} \textit{ is true.}\\ &\textit{event:}\mathit{acceptAuth} \rightsquigarrow \mathit{startAuth} \;\&\; \mathit{notSatisfy} \textit{ is false.}\end{aligned}$$

The correspondence event:*acceptAuth* $\rightsquigarrow$ *startAuth* & *satisfyPolicy* means that, if the process `Verify` executes an event *acceptAuth*, then the processes `Authenticate` and `Judge` have executed the events *startAuth* and *satisfyPolicy* respectively. In other words, if the SP passes the verification and accepts the authentication from a user, then this user has started an authentication session and satisfied the authentication policy before. This means that the SP accepts the authentication from a user who has satisfied the policy, which is immediately corresponding to the statement 1. Similarly, the second failed correspondence means that if the SP accepts the authentication, then the situation that a user has started a session but violated the policy should never happen before. This is to say that the SP would never accept the authentication from users who violate the policy, which is corresponding to statement 2.

Non-frameability A user is framed if he satisfies the authentication policy, but is unable to successfully authenticate himself to an honest SP [8, 9]. Reverse thinking this description, if a system satisfies *non-frameability*, then the situation that a user fails to authenticate to an honest SP but satisfies the authentication policy would never happen. We formalize non-frameability (Definition 5) as the following correspondence.

Definition 5 (Non-frameability). *Given processes* ⟨ `Register`,`Issue`, `Authenticate`,`Verify`,`Judge`⟩, *if correspondence*

$$\textit{event:}\mathsf{revoke} \rightsquigarrow \mathsf{startAuth} \;\&\; \mathsf{satisfyPolicy}$$

is false, non-framebility is satisfied .

This correspondence means that, if SP rejects an authentication, then the situation that a user has started this authentication session and satisfied the authentication policy would never happen, which is corresponding to the statement of non-frameability.

Mis-authentication Resistance Mis-authentication takes place when an unregistered user successfully authenticates himself to an SP. In a system with *mis-authentication resistance*, an SP is assured to accept authentications only from registered users. Analogous to authenticity, we can parse this description into two statements.

1. The statement "a user successfully authenticates to an SP, but he never registered to the issuer before" is false.
2. The statement "if an SP accepts the authentication from a user, then before that, this user has registered with the issuer" is true.

Naturally, these two statements can be formalized into the correspondence properties, as shown in Definition 6 below.

Definition 6 (Mis-authentication Resistance). *if processes* ⟨`Register`,`Issue`, `Authenticate`,`Verify`,`Judge`⟩ *are given, Mis-authentication resistance is satisfied when the following correspondences are held:*

$$\textit{event:}\mathsf{acceptAuth} \rightsquigarrow \mathsf{startAuth} \;\&\; \mathsf{unregistered}\ \textit{is false.}$$
$$\textit{event:}\mathsf{acceptAuth} \rightsquigarrow \mathsf{startAuth} \;\&\; \mathsf{registered}\ \textit{is true.}$$

Privacy The definition of privacy is twofold: *anonymity* and *unlinkability*, which is inspired by the formal definitions in [5].

Anonymity ensures that an adversary cannot see the difference between a system in which the user with a publicly known identity id_0 executes the analyzed processes and the system where id_0 is not present at all. Based on this description, we can formalize the anonymity as follows.

Definition 7 (Anonymity). *Given processes* ⟨Register, Issue, Authenticate, Judge⟩*, anonymity is satisfied if the following equivalence holds:*

$$
\begin{aligned}
&(vid.vp.(\texttt{Register}|(p(cre).!(\texttt{Authenticate}|\texttt{Judge})))) \mid \\
&(let\ id = id_0\ in\ let\ p = int_0\ in \\
&\quad (\texttt{Register}|(p(cre).!(\texttt{Authenticate}|\texttt{Judge})))) \\
&\approx \\
&(!vid.vp.(\texttt{Register}|(p(cre).!(\texttt{Authenticate}|\texttt{Judge}))))
\end{aligned}
$$

Both sides of equivalence are of the same processes except that the left side executes the registration and authentication processes of the user id_0. That is to say, if the equivalence is held, then the adversary cannot tell whether or not the user id_0 has executed the registration and authentication processes. Therefore, the anonymity property is met.

Unlinkability ensures that a system in which the analyzed processes can be executed by a user multiple times looks the same to an adversary that the system in which the analyzed processes can be executed by the user at most once. We formalize the unlinkability as following definition.

Definition 8 (Unlinkability). *Given processes* ⟨Register, Issue, Authenticate, Judge⟩*, unlinkability is satisfied if the following equivalence holds:*

$$
\begin{aligned}
&(!vid.vp.(\texttt{Register}|(p(cre).!(\texttt{Authenticate}|\texttt{Judge})))) \\
&\approx \\
&(!vid.vp.(\texttt{Register}|(p(cre).(\texttt{Authenticate}|\texttt{Judge}))))
\end{aligned}
$$

The difference between two sides locates in the number of times that the authentication has been executed. On condition that this equivalence is satisfied, the adversary cannot distinguish the user executing the authentication multiple times from executing at most once, which is directly corresponding to the definition of unlinkability.

4 Case study: BLACR system

BLACR system is introduced by Au, Kapadia and Susilo [9]. It allows users to anonymously authenticate their identities with an SP directly, while enabling the SP to score users' misbehaviour and deny access from users with insufficient reputation score without the assistance of a TTP. In this section, we model BLACR and automatically verify its security properties using formal analysis tool ProVerif.

4.1 Primitives and Equational Theory

BLACR system employs BBS+ signature scheme, which is proposed by Au *et al.* [10] based on the schemes of Camenisch and Lysyanskaya [27] and of Boneh *et al.* [18]. In this section, we will introduce the primitives described by applied pi calculus and the associated equational theory.

We consider commitment $\texttt{commit}(x, y)$, where x is a message and y is a commitment factor (or blind factor). We also specify an open function together with the signature scheme for permitting signatures on committed values.

We consider BBS+ signature scheme $\texttt{bbssign}(m, \texttt{sk}(s))$, where m is a message to be signed, and s is a key seed to generate signing key pair $(\texttt{sk}(s), \texttt{pk}(s))$. We specify an open function $\texttt{open}(\texttt{bbssign}(\texttt{commit}(x, y),\ \texttt{sk}(s)),\ y)$ for opening the signature of a commitment. Again, we construct a verification function $\texttt{bbsver}(\texttt{open}(\texttt{bbssign}(\texttt{commit}(x, y),\ \texttt{sk}(s)),\ y),\ x,\ \texttt{pk}(s))$ for this signature. Moreover, a message recovery function $\texttt{getmess}(\texttt{open}(\texttt{bbssign}(\texttt{commit}(\text{x},$ $\text{y}),\texttt{sk}(\text{s})),\text{y}))$ is provided to adversary for getting the signing message x.

We construct a zero-knowledge proof as function $\texttt{ZK}_{i,j}(\widetilde{M}, \widetilde{N}, F)$, where $\widetilde{M}$ is private component representing the knowledge to be proved, $\widetilde{N}$ denote the public component and F denotes a formula over those terms.

In summary, we construct a suitable signature Σ and define an equational theory E to capture the operations of cryptographic primitives. The signature can be defined as follows:

$$\begin{aligned}
&\Sigma = \Sigma_{\text{base}} \cup \Sigma_{ZK},\ \text{where}\\
&\Sigma_{\text{base}} = \left\{ \begin{array}{l} \texttt{true,false,hash,exp,and,or,eq,pk,sk,} \\ \texttt{commit,open,bbssign,bbsver,getmess} \end{array} \right\}\\
&\Sigma_{ZK} = \{\texttt{ZK}_{i,j}, \texttt{Ver}_{i,j}, \texttt{Public}_i, \texttt{Formula}, \alpha_i, \beta_j | i, j \in N\}
\end{aligned}$$

For the signature Σ_{base}, functions `true`, `false` are constant symbols; `hash`, `pk`, `sk`, `getmess` are unary functions; `exp`, `land`, `or`, `eq`, `commit`, `open`, `bbssign` are binary functions; `bbsver` is ternary functions. The equation theory E_{base} associated with signature Σ_{base} is defined as follows:

$$\begin{aligned}
&E_{\text{base}} =\\
&\quad \texttt{and}(\texttt{true}, \texttt{true}) = \texttt{true}\\
&\quad \texttt{or}(\texttt{true}, x) = \texttt{true}\\
&\quad \texttt{or}(x, \texttt{true}) = \texttt{true}\\
&\quad \texttt{eq}(x, x) = \texttt{true}\\
&\quad \texttt{bbsver}(\texttt{open}(\texttt{bbssign}(\texttt{commit}(x, y), \texttt{sk}(s)), y), x, \texttt{pk}(s))\\
&= \texttt{true}\\
&\quad \texttt{getmess}(\texttt{open}(\texttt{bbssign}(\texttt{commit}(x, y), \texttt{sk}(s)), y)) = x
\end{aligned}$$

Functions `and`, `or`, `eq` are used for conjunction, disjunction and equality test respectively; `hash` is used for hashing messages; `exp` is used for the exponent operation. The rest functions are used for constructing and verifying BBS+ signature scheme.

The signature Σ_{ZK} and its associated equation theory E_{ZK} can be found in Figure 4 in Section 2, so we will not go into the details here.

4.2 Review of BLACR

In this section, we give a high-level description of the BLACR system. The initialization parameters include: the signing key pair $(\texttt{pk}(s_{iss}), \texttt{sk}(s_{iss}))$ of an

issuer; the unique identity string sid of an SP; the number of categories m and the blacklist of each category with the thresholds TS_i. The registration process proceeds as follows.

1. The issuer sends a random challenge m_{reg} to a user.
2. The user generates a random number y and computes $C_x = \texttt{commit}(x, y)$. Then the user generates a signature proof of knowledge $\Pi_1 = SPK\{(x, y) : C_x = \texttt{commit}(x, y)\}(m_{reg})$. He sends a pair (C_x, Π_1) to the issuer.
3. The issuer computes a blind credential $bcre = \texttt{bbssign}(C_x, \texttt{sk}(s))$ if the verification of Π_1 is successful and then sends $bcre$ to the user.
4. The user opens the blind credential $cre = \texttt{open}(bcre, y)$. He outputs cre as his credential when the verification $\texttt{bbsver}(cre, x, \texttt{pk}(s))$ is true.

After the user obtains a credential cre, he can authentication to the SP multiple times using this credential. The authentication process is presented as follows.

1. The SP sends to the user the lists for each category as well as their corresponding threshold values $\widetilde{TS} = (TS_1, ..., TS_m)$ and a random challenge m_{auth} as well as the policy Pol.
2. The user judges his reputation score s_i of each category by checking if the entries on the corresponding list belong to him. Then he tests if $s_i < TS_i$ so that Pol evaluates to 1.
3. If the test is successful, the user returns to the SP a pair (τ, Π_2, Π_3), where $\tau = (b, t = H(b||sid)^x)$ is the ticket associated with the current authentication session, and (Π_2, Π_3) is a pair of signature proof of knowledges. Π_2 is used to prove that τ is correctly formed with the credential cre: $SPK\{(x, r, cre) : C_x = \texttt{commit}(x, r), \texttt{bbsver}(cre, x, \texttt{pk}(s)) = \texttt{true}, t = \hat{b}^x\}(m_{auth})$, where $\hat{b} = H(b||sid)$; Π_3 is used to prove Pol evaluates to 1: $SPK\{(x, r, s_i) : C_x = \texttt{commit}(x, r),\ C_{s_{ij}} = \texttt{commit}(0)|_{j \notin user}, C_{s_{ij}} = \texttt{commit}(s_{ij})|_{j \in user},\ C_{s_i} = C_{s_{i1}} \cdots C_{s_{iL}}, s_i < TS_i\}(m_{auth})$, where $j \in \{1, ..., L\}$ and L is the length of the corresponding list.
4. The SP verifies the proofs (Π_2, Π_3).

If verification of (Π_2, Π_3) is successful, SP can ensure that the user is a valid one to access the service.

BLACR also realizes a novel approach called express-lane authentication, which can expedite the authentication. To adapt this mechanism, SP should issue a token that is a signature on the aggregated reputation score prior to a time point upon a successful authentication. Then the user can use this token to convince his reputation score in that time instead of proving whether or not an entry belongs to him for every entry in the blacklist. However, using a token disables the SP's capability of unblacklisting since removing entries from blacklist would disable the validity of the token.

$$
\begin{aligned}
&\texttt{Register} = \\
&\quad c(m_{reg}). \\
&\quad \mathit{let}\ x = \texttt{bind}(id)\ \mathit{in} \\
&\quad \nu y. \mathit{let}\ C_x = \texttt{commit}(x, y)\ \mathit{in} \\
&\quad \mathit{let}\ Pi_1 = \texttt{ZK}(x, y; id, C_x, m_{reg}; F_{reg})\ \mathit{in} \\
&\quad \bar{c}\,\langle (C_x, Pi_1)\rangle\,.c(bcre). \\
&\quad \mathit{let}\ cre = \texttt{open}(bcre, y)\ \mathit{in} \\
&\quad \mathit{if}\ \texttt{bbsver}(cre, x, \texttt{pk}(s_{iss}))\ =\ \texttt{true}\ \mathit{then} \\
&\qquad \texttt{event}(\mathit{registered}).!\ \bar{p}\,\langle cre\rangle \\
&\quad \mathit{else} \\
&\qquad \texttt{event}(\mathit{unregistered}) \\
&\texttt{Issue} = \\
&\quad \nu m_{reg}.\bar{c}\,\langle m_{reg}\rangle\,.c((C_x, Pi_1)). \\
&\quad \mathit{if}\ \texttt{public}_2(Pi_1) = C_x\ \mathit{then} \\
&\quad \mathit{if}\ \texttt{public}_3(Pi_1) = m_{reg}\ \mathit{then} \\
&\quad \mathit{if}\ \texttt{Ver}_{2,3}(F_{reg}, Pi_1) = \texttt{true}\ \mathit{then} \\
&\quad \mathit{let}\ id\ = \texttt{public}_1(Pi_1)\ \mathit{in} \\
&\quad \mathit{if}\ \mathit{sybil} = \texttt{true}\ \mathit{then}\ 0\ \mathit{else} \\
&\quad \mathit{let}\ bcre = \texttt{bbssign}(C_x, \texttt{sk}(s_{iss}))\ \mathit{in} \\
&\quad \bar{c}\,\langle bcre\rangle
\end{aligned}
$$

Fig. 5. Process calculus for registration

4.3 Processes for BLACR

We model the registration phase by a pair of processes $\langle \texttt{Register}, \texttt{Issue}\rangle$ presented in Figure 5. We assume that every user has a unique id, which can be a limited resource such as IP, mobile phone number and some others to prevent sybil attack. To model the secret value x bound to limited resource, we present a private function `bind` to construct the secret value $x = \texttt{bind}(id)$. Since function `bind` is private, the adversary cannot reconstruct the secret value x by id. We also assume that there is only one category for blacklist, thus there exists only one threshold value TS.

The user first generates a zero-knowledge proof $\mathit{\Pi}_1 = \texttt{ZK}(x, y; id, C_x, m_{reg}; F_{reg})$ to ensure the ownership of x with the formula $F_{reg} = \texttt{and}(\alpha_1 = \texttt{bind}(\beta_1), \beta_2 = \texttt{commit}(\alpha_1, \alpha_2))$ and sends it to the issuer. The issuer verifies the validity of $\mathit{\Pi}_1$. If the verification is successful, the issuer will check if this user is a sybil. We assume any sybil id has been recorded in a list. To model this mechanism, we introduce a predicate *sybil* in the issuer process. The predicate *sybil* is true if and only if the user id has been marked sybil. For example, we could set $\mathit{sybil} = \texttt{or}(id = \mathit{sybilid}_1, id = \mathit{sybilid}_2)$ if $\mathit{sybilid}_1, \mathit{sybilid}_2$ have been marked sybil. For a valid id, the issuer signs the commitment C_x and sends the blind signature *bcre* to the user.

The user will open the blind signature $bcre$ and get a credential cre. If the verification of cre is true, he will "store" the credential in the private channel p in order to use it in the authentication phase via an input in this channel. Also, the event *registered* will be executed. Otherwise, event *unregistered* is executed.

We model the authentication phase by a pair of processes $\langle$`Authenticate`, `Verify`$\rangle$ presented in Figure 6. To generate zero-knowledge proofs, the user must know his reputation score. However, the calculus of ProVerif cannot afford to handle either the algebraic operations or the state transition. The first limitation hinders the user from computing his reputation score. The second limitation hinders the assigned score of the current authentication session from influencing the next authentication. Fortunately, our security properties focus on "when something happened, then something else would happen as expectation" instead of "how something happened", so we need a trick to model the judgment process. We assume a trusted judgment process outputs the judged score for users. We set the judgment process as follows.

$$\begin{aligned}&\texttt{Judge} = asg(s).\\ &\quad if\ s = TS\ then\ \texttt{event}(\textit{notSatisfy}).\overline{jud}\,\langle TS\rangle\\ &\quad else\ \texttt{event}(\textit{satisfyPolicy}).\overline{jud}\,\langle ltTS\rangle\end{aligned}$$

In a satisfied score judgment, event *satisfyPolicy* is executed and a term $ltTS$ (means less than the threshold value TS) is sent on the private channel jud. Otherwise, event *notSatisfy* is executed and the threshold value TS is sent on the channel jud.

Then the user generates two zero-knowledge proofs: Π_2 with formula $F_{sig} =$ $\texttt{and}(\texttt{and}(\beta_1 = \texttt{commit}(\alpha_1, \alpha_2), \texttt{bbsver}(\ \alpha_3, \alpha_1, \beta_2) = \texttt{true}), \beta_4 = \texttt{exp}(\beta_5, \alpha_1))$ and Π_3 with formula $F_{Pol} = \texttt{and}(\texttt{and}(\beta_1 = \texttt{commit}(\alpha_1, \alpha_2), \beta_2 = \texttt{commit}(\alpha_3, \alpha_4)),$ $\beta_3 = \alpha_3)$. The process executes the event *startAuth* before it outputs proofs $\langle \Pi_2, \Pi_3\rangle$.

The process of SP verifies the proofs $\langle \Pi_2, \Pi_3\rangle$ and executes the event *acceptAuth* when all verifications are passed, otherwise, it executes the event *revoke*. For a successful authentication, this process also "stores" the ticket (b, t) of current authentication by a private channel lt for further assigning reputation score. To model this action, we realize a process `AssignScore` as $lt((b,t)).vs.\overline{pub}\,\langle((b,t),s)\rangle$. It assigns a score to the ticket and outputs this entry in channel asg for the judgment process.

4.4 Experiment Results

In this section, we examine the security properties defined in Section 3 using ProVerif. The processes above will be expressed as specifications of ZKP compiler and then be encoded into the inputs of ProVerif. We describe how to prove these properties and give out the results of ProVerif. The detailed specifications can be found in [38].

$$
\begin{aligned}
&\texttt{Authenticate} = \\
&\quad c(m_{auth}).vr.vb.vr_s \\
&\quad let\ x = \texttt{bind}(id)\ in \\
&\quad let\ C_x = \texttt{commit}(x, r)\ in \\
&\quad let\ h = \texttt{hash}((b, sid))\ in \\
&\quad let\ t = \texttt{exp}(h, x)\ in \\
&\quad let\ Pi_2 = \texttt{ZK}(x, r, cre; C_x, \texttt{pk}(s_{iss}), b, t, h, m_{auth}; F_{sig})\ in \\
&\quad jud(s).let\ C_s = \texttt{commit}(s, r_s)\ in \\
&\quad let\ Pi_3 = \texttt{ZK}(x, r, s, r_s; C_x, C_s, ltTS, m_{auth}; F_{Pol})\ in \\
&\quad \texttt{event}(\mathit{startAuth}).\bar{c}\,\langle(Pi_2, Pi_3)\rangle \\
&\texttt{Verify} = \\
&\quad vm_{auth}.\bar{c}\,\langle(v_{auth})\rangle\,.c((Pi_2, Pi_3)). \\
&\quad if\ \texttt{public}_2(Pi_2) = \texttt{pk}(s_{iss})\ then \\
&\quad if\ \texttt{public}_5(Pi_2) = \texttt{hash}((\texttt{public}_3(Pi_2), sid))\ then \\
&\quad if\ \texttt{public}_6(Pi_2) = m_{auth}\ then \\
&\quad if\ \texttt{Ver}_{3,6}(F_{sig}, Pi_2) = \texttt{true}\ then \\
&\quad let\ C_x = \texttt{public}_1(Pi_2)\ in \\
&\quad let\ b = \texttt{public}_3(Pi_2)\ in \\
&\quad let\ t = \texttt{public}_4(Pi_2)\ in \\
&\quad if\ \texttt{public}_1(Pi_3) = C_x\ then \\
&\quad if\ \texttt{public}_3(Pi_3) = ltTS\ then \\
&\quad if\ \texttt{public}_4(Pi_3) = m_{auth}\ then \\
&\quad if\ \texttt{Ver}_{4,4}(F_{Pol}, Pi_3) = \texttt{true}\ then \\
&\qquad \texttt{event}(\mathit{acceptAuth}).!\ \overline{lt}\,\langle(b, t)\rangle \\
&\quad else \\
&\qquad \texttt{event}(\mathit{revoke})
\end{aligned}
$$

Fig. 6. Process calculus of authentication

Security Properties as Correspondences Security goals *Authenticity*, *Non-frameability* and *Mis-authentication resistance* are expressed by correspondences. To verify these properties, we implement a main process `C-Process` in Figure 7. Note that we also initialize a key pair for the SP since we set $sid = \texttt{pk}(s_{ver})$ to identify the SP for computing the ticket.

Result 1 *Given the main process* `C-Process`, ProVerif *succeeds in proving the correspondence statements defined in Section 3.2. Hence, security properties Authenticity,Non-frameability and Mis-authentication resistance are held under these definitions.*

The running time of the proof is about 10 seconds on an Ubuntu 14.04 located in VMware with Core i3 3.3GHz, 2GiB memory.

```
Process
  vs_iss.vs_ver.vjud.vlt.let c = pub in (
  pub⟨pk(s_iss)⟩ | pub⟨pk(s_ver)⟩ |
  (! issue) | ( ! (Verify | AssignScore)) | ControlUsers
)
```

Fig. 7. Description of `C-Process`

Security Properties as Equivalence Privacy of BLACR is expressed by biprocess. We identify two kinds of privacy: *anonymity* and *unlinkability*. We implement a main process `A-Process` in Figure 8 to capture anonymity. To adapt the definition of anonymity, we use a process `Users` instead of `ControlUsers` to capture arbitrary users (vid) except id_0 executing the processes.

Encoding anonymity in this way, we have the left side of `diff` representing an execution of publicly known id id_0 (as a initial knowledge of adversary), while the right side of `diff` represents an execution of unknown id id_1 (a restrict id). In fact, the right side of `diff` is a case of `Users` (see rule **Red Repl** in semantics). Hence, it directly corresponds to the definition in Section 3.2 and we succeed in reducing the problem of proving anonymity to the diff-equivalence that can be verified by ProVerif.

```
Process
  vs_iss.vs_ver.vjud.vlt.vint_0.vint_1.let c = pub in (
  pub⟨pk(s_iss)⟩ |pub⟨pk(s_ver)⟩ |(!Issue) |Users|
  (let (id, p) = (id_0, int_0) in Register)|
  (vid_1.let (id, p) = (id_1, int_1) in (
    Register|int_0(cre_0).int_1(cre_1).
    let (id, cre) = (diff[id_0, id_1], diff[cre_0, cre_1]) in (
    Authenticate|Judge))
  )
)

where
  Users =!vid.!vp.
          (Register|(p(cre).!(Authenticate|Judge)))
```

Fig. 8. Description of `A-Process`

Result 2 *Given the main process* `A-Process`*,* ProVerif *succeeds in proving the diff-equivalence, therefore, anonymity is satisfied.*

The running time of the proof is almost half an hour in the same virtual machine.

We also implement a main process `U-Process` in Figure 9 to capture unlinkability. Thinking inside this process, we have that the left side of the `diff` representing a user executes the system many time, while the right side represents the users execute the system at most once (The user id_2 is always different for each execution of processes of the user id_1).

For a more intuitive understanding, we identify a copy execution of vid_1 as ${id_1}^1$. Then under this replication (${id_1}^1$), there are arbitrary executions of vid_2 (${id_2}^{11}, {id_2}^{12}..., {id_2}^{1i}, ...$). All the executions identified by ${id_2}^{11}, {id_2}^{12}..., {id_2}^{1i}, ...$ correspond to executions of the same id ${id_1}^1$. Hence, the left side of `diff` may execute the system by a user ${id_1}^1$ multiple times, and the right side only executes different copies identified by $({id_2}^{11}, {id_2}^{12}..., {id_2}^{1i}, ...)$. So we also succeed in reducing the problem of testing unlinkability to diff-equivalence.

```
Process
  vs_iss.vs_ver.vjud.vL.let c = pub in (
  pub⟨pk(s_iss)⟩ |pub⟨pk(s_ver)⟩ |
  (! issue) |Unlinkability
)

where
Unlinkability =
!vid_1.vint_1.(
 (let (id,p) = (id_1,int_1) in Register)|
 (!vid_2.vint_2.(
    (let (id,p) = (id_2,int_2) in Register)|
     (int_1(cre_1).int_2(cre_2).
      let (id,cre) = (diff[id_1,id_2],diff[cre_1,cre_2]) in
      (Authenticate|Judge))
 ) )
)
```

Fig. 9. Description of `U-Process`

Result 3 *Given the main process* `U-Process`*,* ProVerif *succeeds in proving the diff-equivalence, therefore, unlinkability is satisfied.*

The running time of the verification is about twenty minutes in our virtual machine.

$\texttt{Authenticate} =$
$\quad \ldots\ldots$
$\quad c(c, Btk).$
$\quad let\ tk = \texttt{open}(Btk, r_s).$
$\quad if\ \texttt{bbsver}(tk, s, \texttt{pk}(s_{ver})) = \texttt{true}\ then$
$\quad !\overline{p_0}\langle(s, tk)\rangle .!\overline{p_1}\langle(s, tk)\rangle$
$\texttt{Verify} =$
$\quad \ldots\ldots$
$\quad if\ \texttt{Ver}_{4,4}(F_{Pol}, Pi_3) = \texttt{true}\ then$
$\quad\quad \texttt{event}(\mathit{acceptAuth}).!\ \overline{L}\langle(b, t)\rangle .$
$\quad\quad let\ btk = \texttt{bbssign}(C_s, \texttt{sk}(s_{ver}))\ in$
$\quad\quad \bar{c}\langle btk\rangle$
$\quad else\ \ldots\ldots$

Fig. 10. The additions of normal authentication

4.5 Express-lane Case in BLACR

To reward active users[2], BLACR offers express-lane authentication to speed up the authentication process. In the express-lane authentication, an SP additionally signs a credential (a token) on the score of previous time after the verification succeeds. This token will be used in the next authentication.

However, an additional state transition that ProVerif cannot deal with is introduced since a token generated by current authentication must be transferred to the next time for use. Hence we have to bring in another trick to adapt ProVerif. We divide the analysis into two scenarios: the first is the one that a user is honest people and do not misbehaviour purposely, thus always getting a valid token and proceeding as expectation; the second is the one that a user will be revoked and test if BLACR proceeds as expectation. To begin with, we illustrate the models for express-lane authentication.

Firstly, we revise the normal authentication. The registration process stays unchanged. A new registered user always starts with a normal authentication since he does not possess a token. Hence, we should add another procedure for normal authentication to generate the token. The revised process is shown in Figure 10, while we just provide the additions.

In the revised normal authentication, SP additionally signs commitment C_s and sends the signature to the user after verification is successful. Upon receiving this signature, the user opens it and obtains the corresponding token. Then the user outputs a pair (s, tk) in private channels p_0 and p_1, which are used to deliver the token to the next authentication. The reason for outputting this token in two channels will be explained later.

[2] The "active users" has a criterion, which normally is determined by the frequency of access at a fixed time. This is not our concern in the modelling.

$$
\begin{array}{l}
\texttt{ExAuthenticate} = \\
\quad c(m_{Ex}).vr.vb.vr_s.vr_{snew} \\
\quad let\ x = \texttt{bind}(id)\ in \\
\quad let\ C_x = \texttt{commit}(x, r)\ in \\
\quad let\ h = \texttt{hash}((b, sid))\ in \\
\quad let\ t = \texttt{exp}(h, x)\ in \\
\quad let\ Pi_2 = \texttt{ZK}(x, r, cre; C_x, \texttt{pk}(s_{iss}), b, t, h, m_{auth}; F_{sig})\ in \\
\quad let\ C_s = \texttt{commit}(s, r_s)\ in \\
\quad let\ Pi_4 = \texttt{ZK}(x, r, s, r_s, tk; C_x, C_s, \texttt{pk}(s_{ver}), ltTS, \\
\qquad\qquad m_{Ex}; F_{tk})\ in \\
\quad let\ C_{snew} = \texttt{commit}(s_{new}, r_{snew})\ in \\
\quad let\ Pi_3 = \texttt{ZK}(x, r, s_{new}, r_{snew}; C_x, C_{snew}, ltTS, \\
\qquad\qquad TSminus, m_{Ex}; F_{Pol})\ in \\
\quad \texttt{event}(\mathit{startExAuth}).\bar{c}\,\langle(Pi_2, Pi_4, Pi_3)\rangle\,. \\
\quad c(c, btk_{new}). \\
\quad let\ tk_{new} = \texttt{open}(btk_{new}, r_{snew}). \\
\quad if\ \texttt{bbsver}(tk_{new}, s_{new}, \texttt{pk}(s_{ver})) = \texttt{true}\ then \\
\quad !\overline{p_1}\,\langle(s_{new}, tk_{new})\rangle \\
\texttt{ExVerify} = \\
\quad vm_{Ex}.\bar{c}\,\langle m_{Ex}\rangle\,.c((Pi_2, Pi_4, Pi_3)). \\
\quad ...(These\ are\ some\ if...else...\ statements)... \\
\quad if\ \texttt{Ver}_{3,6}(F_{sig}, Pi_2) = \texttt{true}\ then \\
\quad let\ C_x = \texttt{public}_1(Pi_2)\ in \\
\quad let\ b = \texttt{public}_3(Pi_2)\ in \\
\quad let\ t = \texttt{public}_4(Pi_2)\ in \\
\quad jud(snew).let\ C_{snew} = \texttt{public}_2(Pi_3)\ in \\
\quad if\ \texttt{Ver}_{5,5}(F_{tk}, Pi_4) = \texttt{true}\ then \\
\qquad \texttt{event}(\mathit{notWillRevoke}). \\
\qquad if\ \texttt{Ver}_{4,5}(F_{Pol}, Pi_3) = \texttt{true}\ then \\
\qquad\quad \texttt{event}(\mathit{acceptExAuth}).\overline{L}\,\langle(b, t)\rangle\,. \\
\qquad\quad let\ btk_{new} = \texttt{bbssign}(C_{snew}, \texttt{sk}(s_{ver}))\ in \\
\qquad\quad \bar{c}\,\langle btk_{new}\rangle \\
\qquad else\ \texttt{event}(\mathit{revokeEx}) \\
\quad else \\
\qquad \texttt{event}(\mathit{willRevoke}). \\
\qquad if\ \texttt{Ver}_{4,5}(F_{Pol}, Pi_3) = \texttt{true}\ then \\
\qquad\quad ...(the\ same\ as\ above)... \\
\\
where \\
F_{Pol} = \texttt{and}(\texttt{and}(\beta_1 = \texttt{commit}(\alpha_1, \alpha_2), \beta_2 = \texttt{commit}(\alpha_3, \alpha_4)), \\
\qquad\qquad \texttt{or}(\beta_3 = \alpha_3, \beta_4 = \alpha_3)) \\
F_{tk} = \texttt{and}(\texttt{and}(\beta_1 = \texttt{commit}(\alpha_1, \alpha_2), \beta_2 = \texttt{commit}(\alpha_3, \alpha_4)), \\
\qquad\quad \texttt{and}(\texttt{bbsver}(\alpha_5, \alpha_3, \beta_3), \beta_4 = \alpha_3)\ )
\end{array}
$$

Fig. 11. Express-lane authentication

Secondly, we model the express-lane authentication. The processes of express-lane authentication are similar with the normal one, as shown in Figure 11. It adds a zero-knowledge proof Π_4 to prove that the user possesses a valid token. Considering the limitation of space, We omit some "*if...else...*" statements that do not affect comprehension.

We add a predicate $TSminus$ to capture a situation that the reputation score gets close to threshold, and once more punishment will lead to a revocation. We use this predicate to conduct a trick: if a token represents a score that is equal to $TSminus$, then the process executes the event *willRevoke*, otherwise, it is less than the threshold value and event *notWillRevoke* is executed. In both cases, verification of Π_3 is followed. To adapt the predicate $TSminus$, the zero-knowledge proof Π_3 has to be revised as well in that the score $TSminus$ also satisfies the policy. So we add a public component $TSminus$ in Π_3 and modify the formula F_{Pol} as $and(and(...), or(\beta_3 = \alpha_3, \beta_4 = \alpha_3))$.

Scenario 1 In this scenario, we assume the user is behaved as a honest people and always succeed in authenticating himself. The model for express-lane authentication is almost the same as normal-express one except that we need to provide a fresh token for every execution. For instance, we still use main process `C-Process` to verify the correspondence properties defined in Section 3.2. However, the subprocess ControlUsers must be adapted to the following process:

$$\begin{aligned}&\texttt{ControlUsers} = !pub(id).\\ &\quad !\ vp.vp_1.(\texttt{Register}|(p(cre).\\ &\qquad\qquad !(vs.let\ tk = \texttt{bbssign}(s, \texttt{pk}(s_{ver}))\ in\\ &\qquad\qquad\ \texttt{ExAuthenticate}|\texttt{Judge}))).\end{aligned}$$

The newly added item $vs.let\ tk = \texttt{bbssign}(s, \texttt{pk}(s_{ver}))\ in$ is to model a fresh token in `ExAuthentication`. Given the main process `C-Process` in Figure 7, ProVerif also succeeds in proving the correspondence properties.

Scenario 2 In this scenario, the user is dishonest and will be revoked in a certain authentication. To capture this situation, we implement a special main process `S-Process` shown in Figure 12. In this main process, we execute two separate express-lane authentication: the first one issues a token of $TSminus$, and deliver this token to the second one for use. As we expect, the second one should execute event *willRevoke* and does not execute event *NotWillRevoke*. Also, the user has the capability to store all the tokens he processes, so we adopt two private channels to deliver tokens from normal authentication to these two separate express-lane authentications respectively.

Now it is time to model the security property. If BLACR proceeds as expectation, then the event *willRevoke* will be executed in the second express-lane authentication while the event *notWillRevoke* should not be executed.

Given processes $\langle$ `Register`, `Issue`, `Authenticate`, `Verify`, `ExAuthentication`, `ExVerify`$\rangle$, we say that BLACR system proceeds as expectation in this scenario if the following statements hold:

$$\text{event } \textit{willRevoke}(m_{Ex2}) \text{ is executed.}$$

```
Process
  vpriv1.vpriv2.vpriv3.vpriv4.vint1.vint2.vint3.vint4.
  vsiss.vsver.vjud.vL.(pub⟨pk(siss)⟩ | pub⟨pk(sver)⟩ |
  (let c = priv1 in issue) |
  (let (id, c, p) = (id0, priv1, int1) in Register) |
  (let c = priv2 in !Verify) |
  (int1(cre).jud(ltTS).let (id, c, p0, p1) =
          (id1, priv2, int2, int3) in !Authenticate) |
  (let c = priv3 in vmEx1.let mEx = mEx1 in ExVerify) |
  (int1(cre).int2(s, tk).jud(TSminus).let (id, c, p1) =
          (id1, priv3, int3) in ExAuthenticate) |
  (let c = priv4 in vmEx2.let mEx = mEx2 in ExVerify) |
  (int1(cre).int3(s, tk).jud(TS).let (id, c, p1) =
          (id1, priv4, int4) in ExAuthenticate) |
  (!AssignScore)
)
```

Fig. 12. Description of `S-Process`

event *notWillRevoke*(m_{Ex2}) is not executed.

The parameter m_{Ex2} means that we test these statements in the second express-lane authentication. In consequence, ProVerif displays attack derivations for all the statements.

Result 4 *Given the main process* `S-Process`*,* ProVerif *discovers that the events* willRevoke *and* notWillRevoke *are both executed in the second express-lane authentication. As a consequence, we say that the token mechanism of BLACR does not work properly.*

The attack derivation shows that the event *notWillRevoke*(m_{Ex2}) is executed because the token in the normal authentication can be directly used in the second express-lane authentication. Hence, a replay attack can be carried out by a malicious user as follows: in the second express-lane authentication, the user finds his aggregated reputation score does not satisfy the authentication policy. But he still proceeds in this way: he uses a preceding token that is enough to make the aggregate score satisfying the policy. This attack can happen since these tokens do not consist of any labels to distinguish each other.

In general, this attack can be applied to two kinds of scenarios violating the security policy: the first one is that a user can utilize an old token to escape from being revoked; the second one is that a user in possession of a token can conduct an express-lane authentication at any time, regardless of whether he is an active user.

Solution This attack can be fixed by refining the definition of token tk. The token must consist of the timestamp information t and is expressed as a signature on (s, t). In the token-presenting step, the timestamp t must be a public component. To model the revised token mechanism, we add an equation:

$$\begin{aligned}\mathtt{bbsver}(\mathtt{open}(\mathtt{bbssign}((\mathtt{commit}(x,y),\mathtt{commit}(x_1,y_1)),\\ \mathtt{sk}(s)),(y,y_1)),(x,x_1),\mathtt{pk}(s)) = \mathtt{true}\end{aligned}$$

Then SP computes $bcre = \mathtt{bbsign}((C_x, C_t), \mathtt{sk}(s_{ver}))$ and sends it to the user, where $C_x = \mathtt{commit}(x, r)$ and $C_t = \mathtt{commit}(t, r_t)$. The user computes $cre = \mathtt{open}(bcre, (r, rt))$ and obtains the credential cre. Besides, the zero-knowledge proof Π_3 for token representing is revised as

$$\mathtt{ZK}(x, r, s, r_s, tk; C_x, C_s, \mathtt{pk}(s_{ver}), ltTS, m_{Ex}, t; F_{tk}),$$

where

$$\begin{aligned}F_{tk} = \mathtt{and}(\mathtt{and}(\beta_1 = \mathtt{commit}(\alpha_1,\alpha_2), \beta_2 = \mathtt{commit}(\alpha_3,\alpha_4)),\\ \mathtt{and}(\mathtt{bbsver}(\alpha_5,(\alpha_3,\beta_6),\beta_3),\beta_4 = \alpha_3)).\end{aligned}$$

In revised processes, result *not event:*$\mathsf{notWillRevoke}(m_{Ex2})$ is true, which means event $\mathsf{notWillRevoke}(m_{Ex2})$ is not executed.

Discussion In fact, our solution in symbolic representation mode indicates that the fix presented in ExBLACR still does not work properly since the timestamp in the proving process of EXBLACR does not be revealed. In such way, a malicious user can still conduct the replay attack mentioned above, because the SP can just ensure the token tk is correct but can not know the timestamp t corresponding to this token. To reveal the timestamp value, we can not use the standard presenting protocol that BLACR adopts in the implementation of our revised fix. Nevertheless, this standard protocol can be easily refined to meet our need since the timestamp t is public. SP can compute the tth power operations alone to complete a proof of knowledge of a signature, which reflects that the token is used in the right time.

5 Conclusion

This paper presents the definitions of some common security properties for BLAC-like systems in the symbolic model using applied pi calculus. We express these definitions as correspondence properties (*authenticity*, *non-frameability* and *mis-authentication resistance*) and equivalence properties (*anon-ymity* and *unlinkability*) that are suited to verifying by formal analysis tool. As a case study, we verify these properties in BLACR system. The analysis finds a known attack aiming at the token mechanism in the express-lane authentication. We also offer a security revision that makes the token mechanism been successfully proved using ProVerif. This revision indicates that the fix provided by ExBLACR is incorrect.

Actually, our model is of approximate due to the nature of ProVerif. We think some other modelling method can also be under consideration to record state, such as multiset rewriting rules (Tamarin tool [34, 31]), stateful variant of applied pi calculus [6, 23], or other stateful verification framework [30]. Another extension may be lying in research of composing protocols as mentioned in the introduction, which can simplify analysis using ProVerif at a time since too complex inputs will make ProVerif running out of memory. These may be our future work.

Acknowledgements The research presented in this paper is supported by the National Grand Fundamental Research 973 Program of China under Grant No.2013CB338003 and the National Natural Science Foundation of China under Grant Nos. 91118006, 61202414.

References

1. Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM (JACM)*, 52(1):102–146, 2005.
2. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. *ACM SIGPLAN Notices*, 36(3):104–115, 2001.
3. Myrto Arapinis, Vincent Cheval, and Stéphanie Delaune. Verifying privacy-type properties in a modular way. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 95–109. IEEE, 2012.
4. Myrto Arapinis, Vincent Cheval, and Stéphanie Delaune. Composing security protocols: from confidentiality to privacy. In *Principles of Security and Trust*, pages 324–343. Springer, 2015.
5. Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Computer Security Foundations Symposium (CSF), 2010 23rd IEEE*, pages 107–121. IEEE, 2010.
6. Myrto Arapinis, Joshua Phillips, Eike Ritter, and Mark D Ryan. Statverif: Verification of stateful processes. *Journal of Computer Security*, 22(5):743–821, 2014.
7. M Ho Au, Patrick P Tsang, and Apu Kapadia. PEREA: Practical TTP-free revocation of repeatedly misbehaving anonymous users. *ACM Transactions on Information and System Security (TISSEC)*, 14(4):29, 2011.
8. Man Ho Au and Apu Kapadia. PERM: Practical reputation-based blacklisting without TTPs. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 929–940. ACM, 2012.
9. Man Ho Au, Apu Kapadia, and Willy Susilo. BLACR: TTP-free blacklistable anonymous credentials with reputation. In *NDSS Symposium 2012: 19th Network & Distributed System Security Symposium*, pages 1–17. Internet Society, 2012.
10. Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In *Security and Cryptography for Networks*, pages 111–125. Springer, 2006.
11. Michael Backes, Matteo Maffei, and Dominique Unruh. Implementation for the complier from zero-knowledge protocol descriptions into proverif-accepted specifications. `http://www.infsec.cs.uni-sb.de/projects/zk-applied`.
12. Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 202–215. IEEE, 2008.

13. Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *csfw*, page 0082. IEEE, 2001.
14. Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 86–100. IEEE, 2004.
15. Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.
16. Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 331–340. IEEE, 2005.
17. Bruno Blanchet et al. Proverif: Cryptographic protocol verifier in the formal model. `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`.
18. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology–CRYPTO 2004*, pages 41–55. Springer, 2004.
19. Stefan A Brands. *Rethinking public key infrastructures and digital certificates: building in privacy*. Mit Press, 2000.
20. Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145. ACM, 2004.
21. Ernie Brickell, Liqun Chen, and Jiangtao Li. A new direct anonymous attestation scheme from bilinear maps. In *Trusted Computing-Challenges and Applications*, pages 166–178. Springer, 2008.
22. Ernie Brickell and Jiangtao Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 21–30. ACM, 2007.
23. Alessandro Bruni, Sebastian Modersheim, Flemming Nielson, and Hanne Riis Nielson. Set-pi: Set membership p-calculus. In *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*, pages 185–198. IEEE, 2015.
24. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. Cryptology ePrint Archive, Report 2015/1246, 2015. `http://eprint.iacr.org/` (to be apppeared in PKC 2016).
25. Jan Camenisch et al. Specification of the identity mixer cryptographic library, version 2.3. 1, december 7, 2010.
26. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Security in communication networks*, pages 268–289. Springer, 2002.
27. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology–CRYPTO 2004*, pages 56–72. Springer, 2004.
28. Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with proverif. In *Principles of security and trust*, pages 226–246. Springer, 2013.
29. Peter C Johnson, Apu Kapadia, Patrick P Tsang, and Sean W Smith. Nymble: Anonymous IP-address blocking. In *International Workshop on Privacy Enhancing Technologies*, pages 113–133. Springer, 2007.
30. Li Li, Jun Pang, Yang Liu, Jun Sun, and Jin Song Dong. Stateful security protocol verification. *arXiv preprint arXiv:1403.2237*, 2014.
31. Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In *Computer Aided Verification*, pages 696–701. Springer, 2013.
32. Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1. 1. Technical report, revision 3. Technical report, Microsoft Corporation, 2013.

33. Mark D. Ryan and Ben Smyth. Applied pi calculus. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, chapter 6. IOS Press, 2011.
34. Benedikt Schmidt, Sebastian Meier, Cas Cremers, and David Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 78–94. IEEE, 2012.
35. Ben Smyth, Mark Ryan, and Liqun Chen. Formal analysis of anonymity in ECC-based direct anonymous attestation schemes. In *Formal Aspects of Security and Trust*, pages 245–262. Springer, 2011.
36. Ben Smyth, Mark D Ryan, and Liqun Chen. Formal analysis of privacy in direct anonymous attestation schemes. *Science of Computer Programming*, 111:300–317, 2015.
37. Patrick P Tsang, Man Ho Au, Apu Kapadia, and Sean W Smith. BLAC: Revoking repeatedly misbehaving anonymous users without relying on TTPs. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):39, 2010.
38. Weijin Wang. Proverif inputs for analyzing blacr system. `https://github.com/WangWeijin/Formal-analysis-of-BLACR-system`.
39. Weijin Wang, Dengguo Feng, Yu Qin, Jianxiong Shao, Li Xi, and Xiaobo Chu. ExBLACR: Extending BLACR system. In *Information Security and Privacy*, pages 397–412. Springer, 2014.
40. Li Xi and Dengguo Feng. FARB: fast anonymous reputation-based blacklisting without TTPs. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 139–148. ACM, 2014.
41. Li Xi and Dengguo Feng. Formal analysis of DAA-related APIs in TPM 2.0. In *Network and System Security*, pages 421–434. Springer, 2014.
42. Kin Ying Yu, Tsz Hon Yuen, Sherman SM Chow, Siu Ming Yiu, and Lucas CK Hui. PE (AR) 2: Privacy-enhanced anonymous authentication with reputation and revocation. In *European Symposium on Research in Computer Security*, pages 679–696. Springer, 2012.