International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 3– No.9, February 2012 – www.ijais.org

Modelling and Verification of Group Signatures Properties

MBOUPDA MOYO Achille
University of Yaounde I
Faculty of Sciences
Cameroon

ATSA ETOUNDI Roger
University of Yaounde I
Faculty of Sciences
Cameroon

ABSTRACT
A group signature allows any member of a group to sign anonymously and to act on behalf of the group in such a way that only the associated group manager, in case of any dispute, is able to reveal the identity of the author of a transaction. The functioning of the group signature is based on a set of rules that are required to be met at any time by the associated scheme. The state-of-the-art of the group signature research has shown that several schemes have been proposed but none of them satisfy all the fundamental group signature properties such as liveliness. When some of these properties are satisfied, only informal proofs are available. Hence, one cannot claim that these properties are really supported by the associated schemes. This paper contributes in the definition of a group signature scheme by integrating all the required constraints and making it possible to carry out their formal proof. The specification of the scheme is based on the finite state automata techniques and the temporal logic PLTL of CTL* to formally specify the required properties. Finally, the model-checker SPIN is used to verify the consistency of the resulting scheme.

Keywords
Group Signature Scheme, Temporal Logic, Model Verification.

1. INTRODUCTION
The notion of Group signature was introduced by Chaum and Van Heyst [1]. This notion is a very useful tool in applications where the signer's privacy should be protected and in case of abuse an authority can identify the misbehaving user. However, a well-known group signature problem is that it is difficult for the group manager to identify a malicious user since all signatures are anonymous. The group manager obviously cannot afford to open all the group signatures signed, as by so doing, he would compromise the privacy of every signer. To tackle this problem, in practice many group signature schemes have been proposed. In this purpose, M. Abdalla and *al.* in [19] introduced the concept of unique signatures which is of great value from both theoretical and applied perspectives in this domain. Abe and al. in [20, 21] proposed group signatures with efficient concurrent join. A number of unique group signature schemes without random oracles under a variety of security models that extend the standard security models of ordinary group signatures are defined in [22]. Despite the volume of schemes defined in the literature, the formal proof of the associated group signature properties has not yet been carried out based on any of these schemes. This is due to the fact that a generic model of the group signature scheme has not been specified. The existing schemes have been presented based on specific cases. For this end, they do not consider the general aspect of the group signature problem. In order to carry out the proof of these properties, the paper argues that the scheme should first verify two more constraints. Therefore, a scheme obtained from the initial group signature theory cannot be used to formally prove the main properties of a group signature.

Formal specification of application software is becoming more and more common. Communicating finite state machines are suitable for modeling the basic group signature scheme. This motivated our choice of *Promela* as our specification language and *Spin* for checking large state spaces of the system. Although *Promela*'s expressive power is rather low and provides only a few basic data types and primitive constructor types, it offers the possibility of expressing reachability, safety and liveliness requirements by logical assertions or linear temporal logic expressions. The model is further animated by the Spin simulator and verified by the validator [6, 12] with different degrees of precision depending on the problem size. The properties are checked during the simulation or the verification step and if an error occurs it is possible to run the simulator again and look at each state to find the problem.

Only the fundamental group signature scheme of group signatures presented by J. Camenish and M. Stadler [4] informally satisfies most of the required properties of group signatures. A fully specified and verified group signature scheme could find application in a wide variety of fields where information security is a critical issue [5]; examples of information security sensitive domains include: electronic commerce, electronic bank transfer, video conferencing, etc. In this paper, we contribute to research in the area of group signatures by providing a formal specification using finite state automata and a formal verification of group signatures basic properties presented by Camenish and Stadler in [4], which so far have only been informally verified.

The rest of the paper is organized as follows. Section 2 identifies fundamentals of group signatures. Section 3 describes models for the different operations of group signatures using finite state automata techniques. Section 4 formally verifies some properties of our system and provides results generated by the model-checker Spin. Section 5 deals with the conclusion and highlighting some perspectives for future research.

2. FUNDAMENTALS OF GROUP SIGNATURES

A group signature scheme is a digital signature scheme comprised of the following operations [1, 9, 13, 14]:
SETUP: An algorithm for generating the group public key **y** and a group secret key **S**.
JOIN: A protocol between the group manager and a user that results in the user becoming a new group member.
SIGN: A protocol between a group member and a user whereby a group signature on a user supplied message is computed by the group member.
VERIFY: An algorithm for establishing the validity of a group signature giving a group public key and a signed message.
OPEN: An algorithm that, given a signed message and a group secret key, determines the identity of the signer.
According to [9, 10], a group signature scheme must satisfy the following security properties:
Unforgeability: Only group members are able to sign messages on behalf of the group.
Anonymity: Given a signature, identifying the actual signer is computationally hard for everyone except the group manager.
Traceability: The group manager is able to open a signature and identify the actual signer; moreover, a signer cannot prevent the opening of a valid signature.

2.1 Components and Operations

The main protagonist in group signatures schemes are:
- *the group manager*: it's a trusted personality who is in charge of managing the group.
- *group members*: members of a group who can carry out group operations such as signing messages on behalf of the group. After the initial set up of the group which is achieved by the group manager through the SETUP primitive, group members and group manager will undertake JOIN, SIGN, VERIFY and OPEN operations during their stay in the group.

3. OPERATIONS WITH ASSOCIATED AUTOMATA

The proposed model holds on proactive, active and reactive components. A detailed description of each one follows.

3.1 Operation 1 "SETUP"

The group manager computes the following values [9]:
• an RSA modulus **n** and two public exponents $\mathbf{e_1}$, $\mathbf{e_2} > \mathbf{1}$, such that $\mathbf{e_2}$ is relatively prime to **Φ(n),** whereas **Φ** is the Euler **Φ** function;
• two integer $\mathbf{f_1}, \mathbf{f_2} > \mathbf{1}$, whose $\mathbf{e_1}$ - roots and $\mathbf{e_2}$ - roots cannot be computed without knowing the factorization of **n**;
• a cyclic group $\mathbf{G} = \langle \mathbf{g} \rangle_{\mathbf{n}}$ of order **n** in which computing discrete logarithm is infeasible;
• an element $\mathbf{h} \in \mathbf{G}$ whose discrete logarithm to the base **g** must be unknown.
• his public key $\mathbf{y_R} = \mathbf{h}^{\rho}$ for a randomly chosen value $\rho \in Z_n$;
• the group's public key consist of $\mathbf{y} = (\mathbf{n}, \mathbf{e_1}, \mathbf{e_2}, \mathbf{f_1}, \mathbf{f_2}, \mathbf{G}, \mathbf{g}, \mathbf{h}, \mathbf{y_R})$, whereas **ρ** and the factorization of **n** remain the group manager's secret key (also called group secret key).

- **Associated automaton**

Figure 1 bellow provides the automaton that represents the operation.

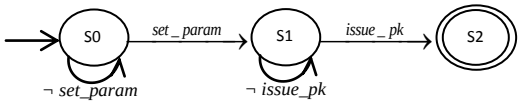


Fig 1: setup automaton: $\mathbf{A_S}$

- Transition elements are boolean variables: ***set_param; issuing_pk***. The symbol (¬) denote the negation.
- Atomic propositions

Here, we denote two atomic propositions.
S1: all components of the group public key have been registered (***set_param*** is set to true).
S2: the group public key has been issued (***issuing_ pk*** is set to true).

- Automaton's formal definition

We obtain the automaton $\mathbf{A_S} = (\mathbf{Q}, \mathbf{E}, \mathbf{T}, \mathbf{Q_0}, \lambda)$ defined as follows:

- the set of states is Q = {S0, S1, S2};
- the set of transition labels is E = {¬set_param; set_param; ¬issuing_pk; issuing_pk};
- the set of transitions is T = {(S0,¬set_param,S0); (S0,set_param,S1); (S1,¬issuing_pk,S1); (S1,issuing_pk, S2)}
- the initial state of automaton is Q_0 = S0
- the application λ which for all states of Q associates the set of elementary properties which is verified in this state is:

$$\lambda = \begin{cases} S0\ a & \varnothing \\ S1\ a & \{S1\} \\ S2\ a & \{S2\} \end{cases}$$

3.2 Operation 2 "JOIN"

Let's call Alice, the person who intends to join the group. Alice. Alice and the group manager both participate in this operation which comprises three steps presented below.

❖ ***first step:***

To become a group member, Alice computes
• $\gamma = r^{e_2}(f_1 y + f_2) \pmod n$ for $\mathbf{r} \in Z_n^*$
• U = E-SKROOTLOG $[\alpha : z = g^{\alpha^{e_1}}]$
• V = E-SKROOTLOG $[\beta : g^{\gamma} = (z^{f_1} g^{f_2})^{\beta^{e_2}}]$
and sends γ, z, U and V to the group manager.

❖ ***second step:***

The group manager verifies whether U and V are correct and sends to Alice the blinded certificate $\gamma = \gamma^{\frac{1}{e_2}} \pmod n$

❖ ***third step:***

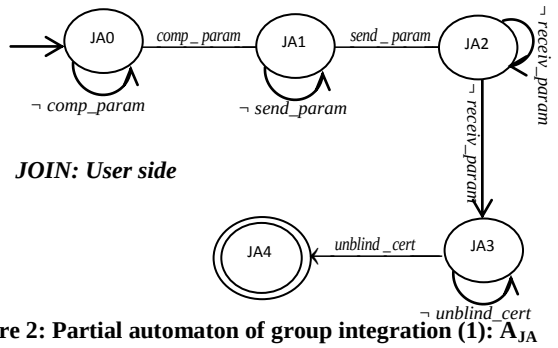
Alice unblinds this certificate and obtains her membership certificate $v = \gamma / r = (f_1 y + f_2)^{\frac{1}{e_2}} \pmod n$

- **Associated automaton**

We will distinguish two automata:
The first one describes the user's (Alice) actions and the second the group manager's actions.

✓ **Alice's case**

This is the automaton that represents the operation.



JOIN: User side

Figure 2: Partial automaton of group integration (1): A_{JA}

- Transition elements are boolean variables: ***comp_param***; ***send_param***; ***receiv_cert***; ***unblind_cert***.
- Atomic propositions

Here, we denote four atomic propositions.
JA1: Alice has computed her member's key (***comp_param*** is set to true).
JA2: Alice has sent a blinded version of her member key to the group manager (***send_param*** is set to true).
JA3: Alice has received a blinded version of her membership certificate from the group manager (***receiv_cert*** is set to true).
JA4: Alice has unblinded her membership certificate (***unblind_cert*** is set to true).

- Automaton's formal definition

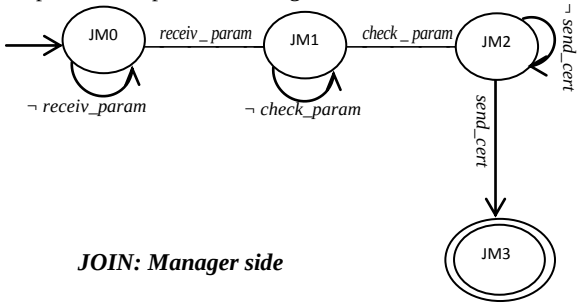
We obtain the automaton A_{JA} = (Q, E, T, Q_0, λ) defined as follows:

- the set of states is Q = {JA0, JA1, JA2, JA3, JA4}
- E = {¬comp_param; comp_param; ¬send_param; send_param; ¬receiv_cert; receiv_cert; ¬unblind cert; unblind_cert;}
- T={(JA0,¬comp_param,JA0); (JA0,comp_param,JA1); (JA1,¬send_param,JA1); (JA1,send_param,JA2); (JA2,¬receiv_cert,JA2); (JA2, receiv_cert,JA3); (JA3,¬unblind_cert,JA3); (JA3,unblind_cert, JA4)}
- the initial state of automaton is Q0 = JA0
- and then,

$$\lambda = \begin{cases} JA0 \text{ a } \varnothing \\ JA1 \text{ a } \{JA1\} \\ JA2 \text{ a } \{JA2\} \\ JA3 \text{ a } \{JA3\} \\ JA4 \text{ a } \{JA4\} \end{cases}$$

✓ **Group manager's case**

The automaton associated with the group manager's set of operations is presented in figure 3 below.



JOIN: Manager side

Fig 3: Partial automaton of group integration (2): A_{JM}

- Transition elements are boolean variables: receiv_param; check_param; send_cert.
- Atomic propositions

JM1: the group manager has received Alice's blinded member key (receiv_param is set to true).
JM2: Alice's signatures of knowledge are correct (check_param is set to true).
JM3: the group manager has sent a blinded version of Alice's membership certificate to her (send_cert is set to true).

- Automaton's formal definition

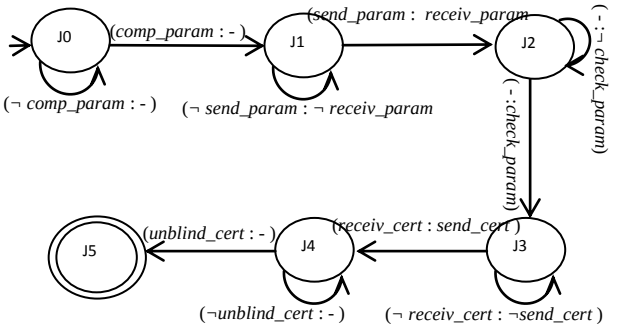
We obtain the automaton AJM = (Q, E, T, Q_0, λ) defined as follows:

- Q = {JM0, JM1, JM2, JM3}
- E={¬receiv_param; receiv_param; ¬check_param; check_param; ¬send_cert; send_cert}
- T={(JM0,¬receiv_param,JM0);(JM0,receiv_param,JM1);(JM1,¬check_param,JM1);(JM1,check_param,JM2);(JM2,¬send_cert,JM2);(JM2,send_cert,JM3)}
- the initial state of automaton is Q_0 = JM0
- and then,

$$\lambda = \begin{cases} JM0 \text{ a } \varnothing \\ JM1 \text{ a } \{JM1\} \\ JM2 \text{ a } \{JM2\} \\ JM3 \text{ a } \{JM3\} \end{cases}$$

3.2.1 Global automaton of "JOIN"

The interaction of the group manager and the user (Alice) in the JOIN primitive compels us to synchronize their automaton [2, 6]. The synchronized product is based on the send / receive message synchronization. In the transition labels, we have separated those which correspond to the send ("send") from those corresponding to the receive ("receive") of the same message.
The synchronization constraint [2] is reduced to the simultaneity of the send and the reception of messages. The set of synchronization is defined as follows:
Sync = {(comp_param, -); (send_param, receiv_param); (-, check_param); (send_cert, receiv_cert); (unblind_cert, -)}, where the symbol "-" means "*do nothing*". After building the cartesian product automaton [2, 3], we extract the synchronized product which is the final automaton of JOIN as shown in figure 4 below.



JOIN

Figure 4: Final automaton JOIN: $A_J = A_{JA} \otimes A_{JM}$

3.3 Scenario 3 "SIGN"

Only the group members intervene in this procedure. To sign a message on behalf of the group, Alice performs the following computations:

- $\not{2}\not{o} = h^r g^y$
- $d = y_R^r$

• V1 = E-SKROOTREP $[(\alpha,\beta): \tilde{z} = h^{\alpha} g^{\beta^{e_1}}](M')$

•V2=E-SKROOTREP $[(\gamma,\delta): \tilde{z}^{f_1} g^{f_2} = h^{\gamma} g^{\gamma^{e_2}}](M')$

•V3=E-SKREP $[(\varepsilon,\zeta): d = R^{\varepsilon} \wedge \tilde{z} = h^{\varepsilon} g^{\zeta}](M')$

The resulting signature on the message **M** consist of ($\tilde{z}$, d, V1, V2, V3) and it is valid if the three signatures of knowledge V1, V2 and V3 are correct.

- Associated automaton

The associated automaton is shown in figure 5.

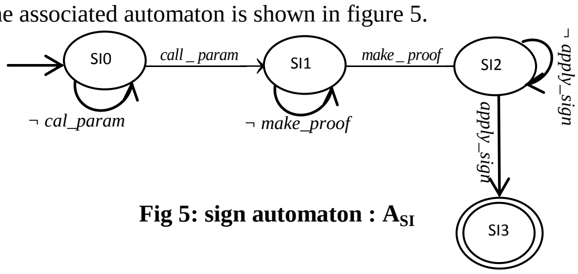


Fig 5: sign automaton : A_{SI}

- Transition elements are boolean variables: cal_param; make_proof; apply_sign.
- Atomic propositions

SI1: Alice has used the group manager's public key to compute the first part of the signature (cal param is set to true).
SI2: Alice has established signatures of knowledge (make_proof is set to true).
SI3: Alice has applied her signature on the massage (apply_sign is set to true).

- Automaton's formal definition

We obtain the automaton A_S = (Q, E, T, Q_0, λ) defined as follows:

- the set of states is Q ={SI0, SI1, SI2,SI3}
- E ={¬cal_param; cal_param; ¬make_proof; make_proof; ¬apply_sign; apply_sign}
- T = {(SI0,¬cal_param,SI0); (SI0,cal_param,SI1); (SI1,¬make_proof,SI1); (SI1,make_proof,SI2); (SI2,¬apply_sign,SI2); (SI2,apply sign,SI3)}
- the initial state of automaton is Q_0 = SI0
- and

$$\lambda = \begin{cases} SI0 \mapsto \varnothing \\ SI1 \mapsto \{SI1\} \\ SI2 \mapsto \{SI2\} \\ SI3 \mapsto \{SI3\} \end{cases}$$

3.4 Operation 4 "OPEN"

Only the group manager is involved here. Given a signature ($\tilde{z}$, d, v1, V2, V3) on a message M, he computes the following:

• $\hat{z} = \tilde{z} d^{1/w}$, which corresponds to the signer's membership key **z**.

• SKREP $[w: \tilde{z} = zd^{w} \wedge h = y_R^{w}](M')$, to prove that **z** is indeed encrypted in $\tilde{z}$ and **d**.

• extracts in **z** the identity of Alice and reveals it.

- Associated automaton

Here is the associated automaton. (figure 6).

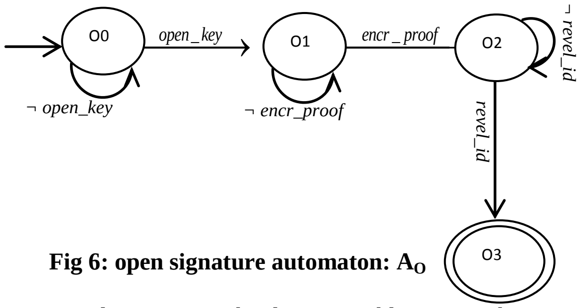


Fig 6: open signature automaton: A_O

- Transition elements are boolean variables: open_key; encr_proof; revel_id.
- Atomic propositions

Here, we denote two atomic propositions.
O1: the group manager has extracted the member key of the signer in the signature (open_key is set to true).
O2: the group manager has proven that the member key is indeed encrypted in the signature (encr_proof is set to true).
O3: the group manager has revealed the identity of the signer (revel_id is set to true).

- Automaton's formal definition

We obtain the automaton A_O = (Q, E, T, Q_0, λ) defined as follows:

- Q ={O0, O1,O2}
- E ={¬open_key; open_key; ¬encr_proof; encr_proof; ¬Revel_id; revel_id}
- T={(O0,¬open_key,O0); (O0,open_key,O1); (O1,¬encr_proof,O1); (O1,encr_proof,O2); (O2,¬revel_id,O2);(O2,revel_id,O3)}
- Q_0 = O0
- and,

$$\lambda = \begin{cases} O0 \mapsto \varnothing \\ O1 \mapsto \{O1\} \\ O2 \mapsto \{O2\} \\ O3 \mapsto \{O3\} \end{cases}$$

3.5 Operation 5 "VERIFY"

Only the group manager intervenes in this operation. He carries out the following tasks [3]:

• check V1 to convince himself that the signer knows the secret key.

• check V2 to convince himself that the signer knows the membership certificate associated to his secret key.

• check V3 to convince himself that the signature could be opened if it necessary.

- Associated automaton

This is the associated automaton (figure 7).

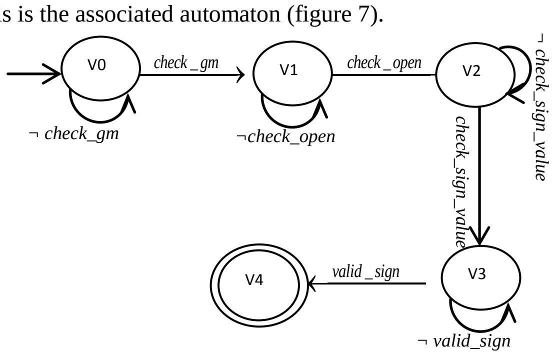


Figure 7: verify automaton: A_V

- Transition elements are boolean variables: check_gm; check_open; check_sign_value; valid_sign.
- Atomic propositions

V1: the group manager is convinced that the first signature of knowledge is correct (check_gm is set to true).
V2: the group manager is convinced that the signature could be opened (check_open is set to true).
V3: the group manager is convinced that the signer is a group member (check_sign value is set to true).
V4: the group manager has validated the signature (valid_sign is set to true).

- Automaton's formal definition

We obtain the automaton A_V = (Q, E, T, Q0, λ) defined as follows:

- Q = {V0, V1, V2, V3,V4}
- E ={¬check_gm; check_gm;¬check_open; check_open;¬check_sign_value; check_sign_value;¬valid_sign;valid_sign}
- T = {(V0,¬check_gm,V0); (V0,check_gm,V1); (V1,¬check_open,V1); (V1,check_open,V2); (V2,¬check_sign_value,(V2,check_sign_value,V3); (V3,¬valid_sign,V3); (V3,valid_sign,V4)}
- the initial state of automaton is Q_0 = V0
- and then,

$$\lambda = \begin{cases} V0\,\text{a}\ \varnothing \\ V1\,\text{a}\ \{V1\} \\ V2\,\text{a}\ \{V2\} \\ V3\,\text{a}\ \{V3\} \end{cases}$$

4. GLOBAL AUTOMATON OF THE SYSTEM

Given that the system is asynchronous, it was split into components that do not interact among themselves. The global automaton will therefore be the cartesian product of automata that represent the various components (SETUP, JOIN, SIGN, VERIFY, OPEN). Hence, a (global) state of the automaton is actually a vector of the various (locals) states of the components.

5. VERIFICATION

The following issues are consigned during verification: preventing major problems such as global deadlock of system, eliminating unspecified receptions and detecting non executable codes at a given moment. Properties verified by the system fall into three major groups presented below. Every property is first of all formalized using LTL (Linear Temporal Logic) of Spin.
The return of Spin is very verbose but rather discreet concerning the truthfulness of a formula. In figure 8 - 12, " never-claim + " in the right diagram indicate that the process of check was launched on the property stated in Formula As typed of the left diagram. Every time that one " - " replace one " + ", it means is that the property was not taken into account, in which case Spin will indicate that there is no error. The met number of errors is indicated to the end of check. If it indicates 0, it means that the property is the true. Otherwise, the shown value will indicate the number of errors. Futhermore, whenever the specication is not satisfied, the ModelChecker will produce a counterexample execution trace that shows why the specification does not hold.

5.1 Reachable properties

A reachable property indicates that a given situation can be reached [3, 6].
They are all atomic propositions that we have identified during the modeling process. To verify them, Spin ensures that states in which these properties are defined are really reachable. The pertinent ones are those which are associated with finals states of automata defined in the previous section.
In SETUP, we have the following proposition:
(S2) *"group public key has been issued"*. This proposition is checked by *assert (issuing pk=1)*.
Where ***assert (boolean variable)*** is a Spin primitive which stops the verification process whenever the Boolean variable is false.
In JOIN, we have the following proposition:
(J5) *"Alice has unblinded her member certificate"*. This proposition is checked by *assert (unblind cert=1)*.
In SIGN, the pertinent proposition is:
(SI3) *"Alice has signed the massage"*. The proposition is checked by *assert (apply sign=1)*.
In VERIFY, we have:
(V4) *"The group manager has validated the signature on the massage"*. This proposition is checked by *assert (valid sign=1)*.
In OPEN, we have:
(O3) *"The group manager has revealed the identity of the signer"*. This proposition is checked by *assert (reveal_id=1)*.

5.2 Safety properties

A safety property means that, under certains conditions, something cannot take place [6]. Examples include *anonymity* and *unforgeability*.

5.2.1 Anonymity

"a group member cannot verify or open a signature on a message". This is formalized in PLTL (Propositional Linear Temporal Logic) as follows:

$$G(member \rightarrow \neg(verify \vee open))$$

The associated automaton generated by Promela in order to run the verification (called *never claim*) is on the left side. (See figure 8)
One can observe in the verification results (on the right side) the large state spaces of the system which had been checked without any error (errors:0).

```
#define member   (_pid > 1 && _pid < 6)
#define verify    (verifie == MESS_VERIFIED)
#define open    (ouvre == MESS_OPENED)
/*
 * Formula As Typed: []  (member -> ! (verify || open))
 * The Never Claim Below Corresponds
 * To The Negated Formula ![]  (member -> ! (verify || open)))
 * (formalizing violations of the original)
*/
never {    /* ![]  (member -> ! (verify || open))) */
T0_init:
if
:: ((((member) && (open)) || ((member) && (verify)))) -> goto accept_all
:: (1) -> goto T0_init
fi;
accept_all:
skip
}
```

```
#ifdef RESULT
(Spin Version 3.5.0)
Bit statespace search for:
never-claim        +
assertion violations+ (if within scope of claim)
acceptance   cycles + (fairness disabled)
invalid endstates- (disabled by never-claim)

State-vector 208 byte, depth reached 1953079, errors: 0
    748 states, stored
3.90609e+06 states, matched
3.90684e+06 transitions (= stored+matched)
4.88195e+06 atomic steps
hash factor: 1399.97 (expected coverage: >= 99.9% on avg.)
(max size 2^20 states)

268.426memory usage (Mbyte)

144.01user 3.11system 3:34.50elapsed 68%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (13516major+65537minor)pagefaults 0swaps
#endif
```

Figure 8: PROPERTY (a): $G(member \rightarrow \neg(verify \vee open))$

5.2.2 *Unforgeability*

"any person who signs a message had previously been registered before as a group member". His PLTL formalization is:

$G(sign \rightarrow F^{-1} join)$ which is equivalent to $(\neg sign \cup join) \vee G\neg sign)$

The *never claim* associated and the verification results are presented in figure 9.

5.3 Liveliness properties

A liveliness property means that, under certains conditions, something will eventually occur [10]. Examples include *belonging*, *traceability* and *feasibility*.

```
#define member    (_pid > 1 && _pid < 6)
#define join   (memcert == MEMCERT)
#define sign   (signe == MESS_SIGNED)
/*
 * Formula As Typed: ( ! sign U join) || []   ! sign
 * The Never Claim Below Corresponds
 * To The Negated Formula !(( ! sign U join) || []   ! sign)
 * (formalizing violations of the original)
 */
never {    /* !(( ! sign U join) || []   ! sign) */
T0_init:
if
:: (! ((join)) && (sign)) -> goto accept_S4
:: (! ((join)) && (sign)) -> goto accept_all
:: (! ((join))) -> goto T0_init
:: (! ((join)) && (sign)) -> goto accept_S13
fi;
accept_S4:
if
:: (! ((join))) -> goto accept_S4
:: (! ((join)) && (sign)) -> goto accept_all
fi;
accept_S13:
if
:: ((sign)) -> goto accept_all
:: (1) -> goto T0_S13
fi;
T0_S13:
if
:: ((sign)) -> goto accept_all
:: (1) -> goto T0_S13
fi;
accept_all:
skip
}
```

```
(Spin Version 3.5.0)

Bit statespace search for:
never-claim        +
assertion violations+ (if within scope of claim)
acceptance   cycles + (fairness disabled)
invalid endstates- (disabled by never-claim)

State-vector 208 byte, depth reached 349, errors: 0
  120199 states, stored
  229172 states, matched
  349371 transitions (= stored+matched)
  410225  atomic steps
hash factor: 8.72359 (best coverage if >100)
(max size 2^20 states)

Stats on memory usage (in Megabytes):
25.963equivalent memory usage for states (stored*(State-vector + overhead))
0.131 memory used for hash-array (-w20)
0.524 memory used for bit stack
252.000memory used for DFS stack (-m9000000)
256.854total actual memory usage

14.66user 2.13system 1:13.64elapsed 22%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (162major+62689minor)pagefaults 0swaps
```

Figure 9: PROPERTY (b): $(\neg sign \cup join) \vee G\neg sign)$

5.3.1 *Belonging*

"one cannot be a group member until he runs join" is formalized in PLTL as follows:

$\neg\ member\ W\ join$

Its equivalence is $\neg member \cup join \vee G\neg member$

Here is the associated *never claim* (figure 10).

```
#define member   (_pid > 1 && _pid < 6)
#define join   (memcert == MEMCERT)
/*
 * Formula As Typed: ( ! member U join) || ! member
 * The Never Claim Below Corresponds
 * To The Negated Formula !(( ! member U join) || ! member)
 * (formalizing violations of the original)
 */
never {    /* !(( ! member U join) || ! member) */
accept_init:
T0_init:
if
:: (! ((join)) && (member)) -> goto accept_S3
:: (! ((join)) && (member)) -> goto accept_all
fi;
accept_S3:
T0_S3:
if
:: (! ((join))) -> goto accept_S3
:: (! ((join)) && (member)) -> goto accept_all
fi;
accept_all:
skip
}
```

```
(Spin Version 3.5.0 )
+ Partial Order Reduction

Full statespace search for:
never-claim         +
assertion violations+ (if within scope of claim)
acceptance   cycles + (fairness disabled)
invalid endstates- (disabled by never-claim)

State-vector 104 byte, depth reached 2547, errors: 0
      178 states, stored
     1457 states, matched
     2678 transitions (= stored+matched)
     2724 atomic steps
hash conflicts: 187 (resolved)
(max size 2^20 states)

256.399memory usage (Mbyte)

0.19user 2.00system 0:59.30elapsed 57%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (156major+62575minor)pagefaults 0swaps
```

Figure 10: PROPERTY (c): $\neg member \cup join \vee G\neg member$

5.3.2 Traceability

"the group manager can sign messages, verify and open signatures on the messages". In PLTL, we have:

$(manager \rightarrow G(sign \vee verify \vee open)$

The associated never claim is presented in figure 11.

5.3.3 Feasibility

"any group member can sign a message on behalf of the group" is translated in PLTL by:

$member \rightarrow \mathbf{F}$ sign

The associated never claim is presented in figure 12.

```
#define member   (_pid > 1 && _pid < 6)
#define join   (memcert == MEMCERT)
#define manager   (_pid == 1)
#define sign   (signe == MESS_SIGNED)
#define verify   ( verifie == MESS_VERIFIED)
#define open   (ouvre == MESS_OPENED)
/*
 * Formula As Typed: manager -> <>  (sign || verify || open)
 * The Never Claim Below Corresponds
 * To The Negated Formula !(manager -> <>  (sign || verify || open))
 * (formalizing violations of the original)
 */
never {    /* !(manager -> <>  (sign || verify || open)) */
accept_init:
T0_init:
if
:: (! ((open)) && ! ((sign)) && ! ((verify)) && (manager)) -> goto accept_S3
fi;
accept_S3:
T0_S3:
if
:: (! ((open)) && ! ((sign)) && ! ((verify))) -> goto accept_S3
fi;
}
```

```
#ifdef RESULT
(Spin Version 3.5.0)
Bit statespace search for:
never-claim         +
assertion violations+ (if within scope of claim)
acceptance   cycles + (fairness disabled)
invalid endstates- (disabled by never-claim)

State-vector 208 byte, depth reached 1995379, errors: 0
     820 states, stored
4.00609e+06 states, matched
4.01684e+06 transitions (= stored+matched)
5.91195e+06 atomic steps
hash factor: 14401.97 (expected coverage: >= 99.9% on avg.)
(max size 2^20 states)

298.321memory usage (Mbyte)

156.01user 3.43system 4:35.50elapsed 77%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (13516major+65537minor)pagefaults 0swaps
#endif
```

Figure 11: PROPERTY (d): $(manager \rightarrow G(sign \vee verify \vee open)$

```
#define member   (_pid > 1 && _pid < 6)
#define sign   (signe == MESS_SIGNED)
/*
 * Formula As Typed: member -> <>  sign
 * The Never Claim Below Corresponds
 * To The Negated Formula !(member -> <>  sign)
 * (formalizing violations of the original)
 */
never {    /* !(member -> <>  sign) */
accept_init:
T0_init:
if
:: (! ((sign)) && (member)) -> goto accept_S3
fi;
accept_S3:
T0_S3:
if
:: (! ((sign))) -> goto accept_S3
fi;
}
```

```
#ifdef RESULT
(Spin Version 3.5.0 )
Bit statespace search for:
never-claim         +
assertion violations+ (if within scope of claim)
acceptance   cycles + (fairness disabled)
invalid endstates- (disabled by never-claim)

State-vector 104 byte, depth reached 225, errors: 0
     405 states, stored
598609 states, matched
570664 transitions (= stored+matched)
675224 atomic steps
hash factor: 2951.4 (expected coverage: >= 99.9% on avg.)
(max size 2^20 states)

173.240 memory usage (Mbyte)

130.01user 3.51system 2:25.50elapsed 75%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (1296major+65596minor)pagefaults 0swaps
#endif
```

Figure 12: PROPERTY (e): $member \rightarrow \mathbf{F}$ **sign**

6. CONCLUSION AND FUTURE WORK

Much work has been done in the group signature field leading to the development of a great number of group signature schemes. However, none of these schemes allows for the carrying out of formal proof of the associated group properties. The defined schemes have been modeled based on specific case studies and do not really take into consideration all the initial outlines properties. In this paper, we have presented a formal model of group signatures specified using finite state automata in order to contribute in the resolution of this difficulty. The proof process has been done incrementally; first we have added and proved two salient properties that are used in verifying the main properties of the group signature defined by D. Chaum and al. The Linear Temporal Logic PLTL which asserts how the behavior of the system evolves over time has been used. Secondly, the SPIN model checker was later used to check whether the abstract model satisfies the properties identified and formalized using PLTL temporal logic. These properties include: reachable properties, safety properties and liveliness properties. By applying each property in turn as well as the model of the system to the SPIN model-checker, we formally verified that the defined model satisfied all the properties listed by D. Chaum. During the verification process, liveliness properties such as "belonging" and "feasibility" impose themselves as conditions of the achievement of the "traceability" required to be one of the fundamental group signature properties. As future work, we plan to implement the defined group signature scheme, and extend the model in order to carry out verification of a group signature scheme in which the group management is shared among its members such that every member is involved in all management transactions.

7. REFERENCES

[1] D. Chaum: "Blind signature systems". In advances in cryptology-CRYPTO, 1983, pages 153. Plenum Press, 1984.

[2] G. Berry, P. Couronne and G. Gonthier: "Synchronous Programming of Reactive Systems: An introduction to ESTEREL". Proc. 1st Franco-Japanese Symp. on programming of Future Generation Computers, 1986, Tokyo. pp.35-56.

[3] W. Thomas: "Automata on Infinite Objects". Handbook on Theoretical Computer Science, J. Van Leeuwen, ed, pp. 133-187, Elsevier Science, 1990.

[4] D. Chaum and E. Van Heyst: "Group signatures". In advances in cryptology-EUROCRYPT 1991, volume 547 of Lecture notes in computer Science, pages 257-265. Springer-Verlag, 1991.

[5] G. J. Holzmann: "Design and Validation of Computers Protocols". Englewood Cli s, N.J. Prentice Hall, 1991.

[6] Z. Manna, A. Pnueli: "The Temporal Logic of Reactive and Concurrent Systems Programs". Springer-Verlag, 1991.

[7] G. J. Holzmann: "Basic Spin Manual". AT&Bell Laboratories, Murray Hill, New Jersey. 1995.

[8] J. Magnier: "Représentation symbolique et Vérification formelle de machines séquentielles". State Thesis University of Montpellier II, France, 1996.

[9] J. Camenisch: "Efficient and generalized group signatures". In advances in cryptology-EUROCRYPT'97, volume 1233 of Lecture notes in computer Science, pages 465-479. Springer-Verlag, 1997.

[10] Jan Camenisch and Markus Stadler: "Efficient Group Signatures Schemes for large Groups". In advances in cryptology-CRYPTO, 1997, volume 1294 of Lecture notes in computer Science, pages 410-424. Springer-Verlag, 1997.

[11] B. Berard, M. Bidoit, F. Laroussinie, A. Petit, P. Schnoebelen: "Vérification de logiciels – Techniques et outils du model-checking". Vuibert, Paris, 1999.

[12] G. Ateniese and G. Tsudik: "Some Open Issues and New Directions in Group Signatures". In cryptography'99, 1999.

[13] Dan Boneh, Xavier BoyenetHovav Shacham: "Short Group Signatures" Advances in Cryptology – CRYPTO 2004, Volume 3152/2004, 227-242

[14] Jan Camenische and Jens Groth: "Group Signatures: Better Efficiency and New Theoretical Aspects" Security in Communication Networks, 2005, Volume 3352/2005, 120-133.

[15] Jens Groth: "Fully Anonymous Group Signatures without Random Oracles" Advances in Cryptology – ASIACRYPT, Volume 4833/2007, 164-180.

[16] Xiaojun Wen, Yuan Tian, Liping Ji1 and Xiamu Niu "A group signature scheme based on quantum teleportation" Physica Scripta; Volume 81; Number 5; May 2010.

[17] S. Dov Gordon and Jonathan Katz and Vinod Vaikuntanathan "A Group Signature Scheme from Lattice Assumptions" Cryptology ePrint Archive, Version: 20110208:170219, last revised 8 Feb 2011.

[18] G. J. Holzmann: "Basic Spin Manual". AT&Bell Laboratories, Murray Hill, New Jersey.

[19] M. Abdalla, D. Catalano, and D. Fiore. "Veritable random functions from identity-based key encapsulation. EUROCRYPT 2009, LNCS vol. 5479, Springer, pp. 554-571, 2009.

[20] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. "Structure-perserving signatures and commitments to group elements". CRYPTO 2010, LNCS vol. 6223, Springer, pp. 209-236, 2010.

[21] M. Abe, J. Groth, K. Haralambiev, and M. Ohkubo. "Optimal structure-preserving signatures in asymmetric bilinear groups". CRYPTO 2011, LNCS vol. 6841, Springer, pp. 649-666, 2011.

[22] Matthew Franklin, Haibin Zhang "Unique Group Signatures", ESORICS 2012.