A formal approach toward authenticated authorization without identification

Lucas D. Witt

Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation

http://lib.dr.iastate.edu/rtd/15313

This Thesis is brought to you for free and open access by Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
A formal approach toward authenticated authorization without identification

by

Lucas D Witt

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Assurance

Program of Study Committee:
Johnny S. Wong, Co-major Professor
Samik Basu, Co-major Professor
Doug Jacobson
Wenshang Zheng

Iowa State University
Ames, Iowa
2008

Copyright © Lucas D Witt, 2008. All rights reserved.
# TABLE OF CONTENTS

| LIST OF TABLES .................................................................................. | v |
| LIST OF FIGURES ............................................................................... | vi |
| CHAPTER 1. INTRODUCTION .................................................................. | 1 |
| CHAPTER 2. BACKGROUND ................................................................... | 3 |
| 2.1 Ad hoc networks ......................................................................... | 3 |
| 2.2 Model checking .......................................................................... | 4 |
| 2.3 Authentication .......................................................................... | 4 |
| 2.4 Authorization ........................................................................... | 4 |
| 2.5 Methods used ........................................................................... | 5 |
| CHAPTER 3. RELATED WORK ............................................................. | 7 |
| 3.1 Authentication, authorization, and identification ..................... | 7 |
| 3.2 Model checking .......................................................................... | 8 |
| CHAPTER 4. OUR PROPOSED SCHEME ............................................... | 9 |
| 4.1 Objectives ................................................................................. | 9 |
| 4.2 Architecture ............................................................................. | 11 |
| 4.3 Assumptions ............................................................................. | 12 |
| 4.4 Protocol notation ...................................................................... | 13 |
| 4.5 The Protocol ............................................................................ | 14 |
| 4.5.1 Initialization ........................................................................ | 14 |
| 4.5.2 Phase I: chain validation ...................................................... | 14 |
| 4.5.3 Phase II: group registration .................................................. | 15 |
4.5.4 Phase III: message sending .......................... 17
4.5.5 Phase IV: sybil speculation .......................... 18

CHAPTER 5. FORMAL VERIFICATION - MODEL CHECKING .......... 21
  5.1 Modeling decisions .................................. 21
  5.2 Model description .................................. 22
    5.2.1 Syntax and semantics ............................ 22
    5.2.2 Code segment 1 .................................. 23
    5.2.3 Code segment 2 .................................. 23
    5.2.4 Modeling multiple nodes ......................... 24
  5.3 Property verification ................................ 25
    5.3.1 LTL syntax ....................................... 26
    5.3.2 Claim verification ................................ 26
    5.3.3 Sanity checks .................................... 30
    5.3.4 Model checking results ........................... 33
    5.3.5 Validity of claims considering infinite process instances ........................................ 34

CHAPTER 6. ANALYSIS AND DISCUSSION ......................... 36
  6.1 Informal observations ................................ 36
    6.1.1 Group size assessment ............................ 36
    6.1.2 Storage requirements .............................. 37
    6.1.3 Computation requirements ......................... 37
    6.1.4 Communication requirements ...................... 37
  6.2 Security ............................................. 38
    6.2.1 Sybil attacks .................................... 38
    6.2.2 Replay attacks ................................... 38
    6.2.3 Denial-of-service attacks ......................... 39
    6.2.4 Loquacious ambassador ............................ 39
  6.3 Maintenance .......................................... 39

CHAPTER 7. CONCLUSION AND FUTURE WORK .................... 41


LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>protocol notation</td>
<td>13</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>results of model checking segment 1</td>
<td>33</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>results of model checking segment 2</td>
<td>34</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>network environment</td>
<td>11</td>
</tr>
<tr>
<td>4.2</td>
<td>phase 1: chain validation</td>
<td>16</td>
</tr>
<tr>
<td>4.3</td>
<td>phase 2: group registration</td>
<td>17</td>
</tr>
<tr>
<td>4.4</td>
<td>phase 3: message sending</td>
<td>19</td>
</tr>
<tr>
<td>4.5</td>
<td>phase 4: sybil resolution</td>
<td>20</td>
</tr>
</tbody>
</table>
CHAPTER 1. INTRODUCTION

Anonymity is a valued commodity on the Internet many uses take advantage of. Proof of demand lies in the popularity of services such as anonymous chat rooms, Tor networks[(11)], and anonymous email services. A negative stigma or even suspicion may quell desire to use these services, but there are many legitimate purposes for withholding identity. For example, a corporate whistle blower may wish to publicly expose surreptitious law violations. That individual may also wish to remain anonymous for fear of media attention or reprimand in the future. Similarly, consider that public elections hold each individual’s vote privacy of utmost importance. Every election participant must prove his or her authorization to vote with some means of identification, such as a driver’s license. Unfortunately, anonymity and credibility are essentially mutually exclusive. A practical method to determine request creditability without also inheriting the capability to identify the exclusive source of a request is not in common practice. Many resent authorization with biometrics since it requires some uniquely defining, physical feature such as fingerprint.

Previous research efforts have proposed methods to ensure anonymous communication, but most sacrifice either authorization or privacy. Reflect back on the election scenario, a situation where quantity of messages is important. Every voter is only allowed one vote, else they could hold unbalanced impact on the outcome. Multiple scenarios are sensitive to some quantity of messages just as elections are. User accountability becomes important in an environment where decisions are based on observation of exterior sources. In this paper, we propose a scheme that allows a message sender to prove his or her creditability without also providing any participant capability to identify the source. We call this “authorization without identification”. In addition, the scheme resists user attempt to violate some enforced message quota and also disjoints
correlation of independent message to a similar source. We focus on the ad hoc network environment where an infrastructure to provide traditional transmission-contingent authentication may be absent. Our effort accomplishes four progressive contributions:

- We propose and evaluate a protocol that provides authenticated authorization without personal identification on a per message basis.

- We evaluate protocol resistance to sybil attacks and association among independent message.

- We implement a formal model of our protocol in Promela.

- We utilize model checking techniques to verify correctness of properties we claim our protocol holds.

The paper continues with background information regarding the focus of our effort. Chapter 3 presents the reader with an overview of previous work that served as reference and motivation for our solution. Chapter 4 follows with a procedural explanation of our protocol. Chapter 5 summarizes our use of model checking, and chapter 6 contains a discussion of properties not formally verified. Chapter 7 concludes our presentation with conclusions and future work. Appendix A contains a full listing of code.
CHAPTER 2. BACKGROUND

In this section, we first provide the reader background information regarding an ad hoc network environment and summarize the practice of model checking. Second, we define the role and importance of authentication and authorization. Lastly, we formally define the objectives of our protocol and explain the methods used to fulfill these objectives.

2.1 Ad hoc networks

The progression of condensing technology into ever more discrete devices has initiated and expanded the realm of mobile computing. Being untethered to specific locale amplifies the demand for less restrictive communication infrastructure\[(8)\]. Whereas a traditional wireless network relays transmission directly though some adjacent, wired base station, an ad hoc network performs multi-hop routing to distribute access to a network backbone layer that communicates directly with a structured, wire connection. Mobile devices can rely on this peer-to-peer communication to provide a simple means for global communication in isolated regions. More specific ad hoc networks include sensor networks and vehicular ad hoc networks(VAHNETs). A wireless sensor network consists of discrete microprocessors with limited computational resources that collect data of interest from the surrounding environment\[(8)\]. Wireless sensors may be subject to isolated and hostile conditions with little means to base security practices upon trusted peers. Vehicular ad hoc networks perform inter-vehicle communication with support from roadside network infrastructure. A vehicle may devise real-time suggestions based on consensus messages from surrounding vehicles, such as route alteration due to unfavorable traffic conditions\[(12)\].
2.2 Model checking

Human nature tends to consider the probability of a certain situation when designing. Model checking considers possibility rather than probability\(\text{(4)}\). Model checking is a discipline within the field of formal methods used to verify the absence of logical errors in a system\(\text{(6)}\). Initially, a system is described as a finite state model. A model checker is a software tool that decides where a property holds in the given description of a model\(\text{(9)}\). It performs formal verification of defined design properties with a systematic, exhaustive search over the derived state space. Properties are expressed in temporal logic. Two fundamental properties consider safety, which specify an unwanted state is never reached, and liveness properties, which specify a desired state is eventually reached\(\text{(6)}\). Some measure of abstraction permeates models in attempt to reduce model complexity and state quantity. With formal verification we can eradicate any concern some undesired instance may occur.

2.3 Authentication

Users commonly consider authentication measures hindering rather than a practice to ensure some controlled environment for certified participants. It ensures all admitted persons merit rightful access based on identity and enforces some standard for participation. Both are vital to uphold network integrity. Authentication is more efficient than restoration in the same way preventive action is much more efficient than recovery. In our scheme, authentication is the process of attempting to confirm the digital identity of a sender of communication is genuine\(\text{(13)}\). According to this definition, authentication is not possible without identification. Authentication ensures an individual is who he or she claims to be, but does not consider access rights of the individual.

2.4 Authorization

In security systems, authorization is distinct from, but reliant upon, authentication. Authorization is the act of determining whether an entity has the right to do what it requests.
The result is either a granted or denied request or request, based on a set of permissions an authorized entity is allowed. Authorization is dependent upon the binding of these permissions to some credential. Non-arbitrary authorization cannot occur without authentication, requiring us to separate this credential from an authorized identity to uphold anonymity.

2.5 Methods used

**Pseudonym:** A pseudonym is an alias, presented as a substitute identifier in an attempt to keep some distinct identifier private.

**Hash chain:** One-way hash function $h(x)$ is a computationally efficient method to transform some input into a fixed-size output. A chain is derived by generating an initial input value and repeatedly hashing the output. This results in a sequence of values, each the hash of the preceding value. Thus, $h(h(h(h(\alpha))))$ yields a sequence $a^0, p^1, p^2, p^3, ..., p^t$, where $t$ is the number of times the function is performed. These values can be revealed in reverse order of generation to indicate knowledge of the initial input, $\alpha^0$: the *chain anchor*. Once an entity proves knowledge of $p^j$, a recipient cognizant of *chain tail* $p^t$ can repeatedly hash $p^j$ to eventually verify that same entity’s knowledge of any $p^x$ between $p^{j+1}$ and $p^t$. The use of a hash chain allows for convenient authorization from a single interaction at the cost of using linkable values among hashes of a common chain. We refer to each value in a hash chain a *link*, denoted by $L$, alluding to the interlocking sequence of links that compose a physical chain. Once a chain tail is authenticated, the user selects the previous hash chain value $L^{t-1}$, so $L^t$ is never used, else the bound identity is exposed.

**Obfuscation:** Encryption is commonly utilized on the Internet to render content obscure or unclear to any observer who is not authorized to read it[(13)]. The goal is not to hide act of communication, but to render the data being communicated unintelligible. Despite its commonality, encryption is not always the simplest form of obfuscation. Our more computationally efficient method utilizes the properties of hash functions to mask the fact that $h(x) = z$. Simple addition renders a hash function inconsistent: $h(x+y) \neq z$. 
Blind signature: The blind signature scheme is a modification of the digital signature scheme in which message content retains a signature without being exposed the signer. Suppose Alice obfuscates, or “blinds”, a value and sends it with proof of identity to Bob. Bob will sign the content with his private key based upon verified identification from Alice, even though he cannot determine the original value. Upon receiving the blinded and signed credential in response, Alice can unblind the credential without disturbing the signature to reveal the original value with Bob's signature. Blind signatures separate the identity of an owner from his or her revealed content. When Alice sends the signed, unblinded credential back, Bob actually cannot determine Alice as the sender because he cannot recognize it. However, Bob can conclude he received the credential from a source he recognized and authorized in some previous encounter. Blind signatures are commonly used in digital cash schemes and voting protocols, when ensuring private but authentic communication is vital. Blind signature schemes can be implemented using a number of digital signature schemes, such as RSA[3].
CHAPTER 3. RELATED WORK

Previous research investigating the use of anonymous credentials is somewhat sparse and rather diverse. Chaum’s development of both the dining cryptographers problem and zero-knowledge proofs[1] initiated the pursuit. He also derived a substantial building block with the blind signature scheme[2]. We first look at the efforts to address authorized, anonymous action and then introduce previous work integrating security with model checking.

3.1 Authentication, authorization, and identification

Authentication must be exercised at some level in a network requiring access control, so it is up to security-minded to separate authorization from authentication to provide controlled, yet anonymous communication. The crucial basis for our protocol relies on blind signatures when a user authenticates with a third party, much in the way [3] does. Our major difference lies in their publication a service provider relies on the third-party to authorize each session between service and users. We focus on individual transmissions in which constant external administration would consume considerable resources.

Maintaining anonymity while authorizing multiple independent requests presents a challenge. Chaum introduced the use of pseudonyms to preserve anonymity, and [15] has summarized uses of pseudonyms. They are used in [7] and [12] to utilize pseudonyms for inconsistent message source. Use of multiple identifiers holds the potential for a user to appear to be multiple, unique or logical sources while in actuality being a single source. Such behavior is called a sybil attack. [7] presents a solution to sybil attacks in a vehicular ad-hoc network using two-tiered oversight by passive devices. However, this paper also introduces an omniscient authority that eventually threatens all absolute privacy.
3.2 Model checking

The expanding use of model checking and greater concern for security has resulted in several efforts attempting to provide cryptographic considerations in model checking. Several model checkers were developed with explicit cryptographic consideration, such as Casper[(16)] and CSP[(17)]. However, the flexibility of model checking allows one to abstract these also. [(5)] and [(18)] highlight an implementation of how they modeled the Needham-Schroeder protocol in SPIN. We adhere to the convention from the former in our model.
CHAPTER 4. OUR PROPOSED SCHEME

In this chapter we formally present our protocol, beginning with objectives and notable architecture and follow with assumptions made to reduce our scope. It continues with a concise summary of notation and concludes with a procedural description of the protocol.

4.1 Objectives

In our protocol we consider a ad hoc network participant wants to anonymously communicate with other nodes. These recipient nodes wish to know sender legitimacy and hold each accountable to a certain message quota. This simplified scenario leads us to define the following objectives:

**Objective 1.** Privacy-preserving authentication

A user can prove his or her identity without revealing some complementary identifier.

In order to implement a secure environment with authentication, participants must be able confirm the identity of other participants. User verification is necessary to determine whether a request can be fulfilled in accordance with defined permissions or policies. However, to fulfill our objective, an authentication process must take place without exposing the relationship between authenticated users and their unique identifiers. A user must somehow conceal either their credentials or identity when presenting both simultaneously. A blind signature scheme allows an authenticator to authorize a credential based on peer identity without being able to identify the credential.

**Objective 2.** Authorization of anonymous senders
A message source is able to prove permissions to external parties without exposing it’s identity as the source.

A user may wish to transmit a message or perform some action without revealing itself as the source in some instances. Authorization confirms that some requester has permission to take the action they seek. Our objective here is to allow a user to be authorized without disclosing his or her identity. With support from the previous objective, a recipient is able to discern the validity of an identifier without being able to distinguish the owner.

**Objective 3.** Message dissociation

Observers cannot correlate independent messages to a common source

The first two objectives require the source of a message cannot be identified. An individual may wish to hide a sequence of specific actions that trigger alarm even when they cannot be identified as the common source. For example, learning an undetermined relative has purchased streamers, balloons, and a cake may raise suspicion someone is devising a surprise party. Even if an observer is not able to determine the absolute identity of a message source, they could be aware and may prepare for what was planned to be unexpected. By determining message identifiers are from a single origin, an adversary can infer certain actions are likely to take place. Thus we find it desirable to hide any association between actions by making the source of any given message indistinguishable from the source of a different message.

**Objective 4.** Detects attempted sybil attacks

Pseudonyms can be a double-edged sword for identity management. Unlimited, disassociated identifiers allow an individual to masquerade as multiple parties, but pose a problem when some sort of user accountability or reputation profile is required. A sybil attack occurs when a malicious entity pretends to be multiple other entities\(^{(\text{7})}\). One disorderly source could repeat messages with unique identifiers to create the illusion of correspondence among multiple neighbors. An participant may gain disproportional influence in an environment considerate of complied, recurring message.
**Objective 5.** Formal verification

We use a model checking tool to verify defined protocol properties without physical implementation. Formal verification is a practice to prove the validity of a scheme. It provides more concrete proof than optimistic reliance on intuitive claims of protocol capability.

**Objective 6.** Scalable to a wireless environment

One advantage of an ad hoc network is its ability to adapt to changing topology, membership, and specification with minimal overhead. Our protocol must be able to gracefully accommodate these features without violating the preceding objectives.

### 4.2 Architecture

![Network Environment Diagram](image)

**Figure 4.1** network environment

User(U) - A user is defined as a generic participant with no privileges, responsibilities, or restrictions beyond the ability to send and read transmissions on a network as any generic communications device currently can.

Group(g) - A group is a domain of voluntary users under the jurisdiction of a single, approved ambassador (see Ambassador below). A group is identified by tuple \(<\text{gid}, L^0, L^t_g>\).
Ambassador(A) - An ambassador is a user who holds authority of a group to which his identity is bound. The ambassador role becomes necessary to resolve speculation of sybil attacks and manage group organization and maintenance. We are not concerned with how a user is promoted to ambassador status, but he ideally would be considered a trusted party prior to promotion. Ambassadors do not initiate communication messages in our model. In reality such limitation may make users reluctant to fulfill such a role; so each ambassador would be required to register with an external group for sending non-administrative messages.

Public Key Infrastructure(PKI) – Our model relies on public-key infrastructure to authenticate users as well as ensure privacy of sensitive information. PKI holds widespread use on the Internet today and requires no special augmentations to accommodate. Essentially, PKI relies on a trusted third-party to accommodate communication between parties not sharing a symmetric encryption key[(10)].

4.3 Assumptions

**Assumption 1**: A pseudonym is only valid for use once per chain.

Each pseudonym expires after a single use. After the intended recipient receives it, all duplicates are rejected.

**Assumption 2**: Every user is aware of all expired pseudonyms.

In a traditional wireless network each node observes every transmission within range. If it determines itself as the intended recipient it processes the packet content, otherwise it discards the packet. Our protocol requires that when a node checks the intended destination, it also updates knowledge of the most recently used pseudonym. Otherwise, uninformed users are not aware which pseudonyms have expired and will erroneously accept any message with a replayed, but expired, pseudonym.

**Assumption 3**: A user can only be member of any group, but only one at a time.

We disallow a user simultaneous group memberships to prevent him or her from mitigating any sybil detection. Sending repeated messages using links from different group chains would
prevent any suspicion of a sybil attack by the receiver. All messages would appear to be from different groups. In practice, a collaboration among ambassadors could collectively check that a proposing member with public key $\pi$ is not already registered.

**Assumption 4**: Every ambassador is a trusted party.

For simplicity we consider every ambassador to be an authentic, reputable third party who candidly performs all requested duties. We briefly consider a less ideal ambassador in section 6.2.4

**Assumption 5**: Hash collisions do not occur.

A hash collision occurs when the hash unique values $x$ and $y$ result in the same output, such that $h(x)=h(y)$. The probability of finding a collision is negligible and in worst case would result in incorrect speculation of a sybil attack[(14)].

### 4.4 Protocol notation

Table 4.1 contains a summary of the notation used in the following protocol description.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_i$</td>
<td>user $i$</td>
</tr>
<tr>
<td>$cert_U$</td>
<td>certificate of user $U$</td>
</tr>
<tr>
<td>$L_j^U$</td>
<td>unused link of user hash chain, with index $j$</td>
</tr>
<tr>
<td>$k_U$</td>
<td>symmetric key of user $U$</td>
</tr>
<tr>
<td>$L_j^g$</td>
<td>unused link of group hash chain, with index $j$</td>
</tr>
<tr>
<td>$A_i$</td>
<td>ambassador $i$</td>
</tr>
<tr>
<td>$n_X$</td>
<td>nonce of participant $X$</td>
</tr>
<tr>
<td>$h(x)$</td>
<td>hash of value $x$</td>
</tr>
<tr>
<td>$pk_X$</td>
<td>public key of participant $X$</td>
</tr>
<tr>
<td>$sk_X$</td>
<td>private key of participant $X$</td>
</tr>
<tr>
<td>$R_i$</td>
<td>recipient $i$</td>
</tr>
<tr>
<td>$M$</td>
<td>message content</td>
</tr>
<tr>
<td>$Bu$</td>
<td>blinded and unsigned link</td>
</tr>
<tr>
<td>$Cs$</td>
<td>unblinded and signed credential</td>
</tr>
<tr>
<td>$Bs$</td>
<td>blinded and signed link</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>(output)</td>
</tr>
<tr>
<td>$s_U$</td>
<td>random salt value for link obfuscation</td>
</tr>
<tr>
<td>$decrypt(k_X): y$</td>
<td>decrypt $y$ with $k_X$</td>
</tr>
</tbody>
</table>
4.5 The Protocol

In our scheme, each active network member has two sets of pseudonyms. These pseudonyms are not merely identifiers, but also means to prove sender authorization. One set of pseudonyms is issued by a group each member is required to join; the other set is computed individually. Both sets are derived from a hash chain. A member uses a valid group credential and also sends a unique personal identifier that cannot be linked back to reveal their identity. The protocol is separated into four distinct phases, preceded by administrative initialization we briefly mention. Numbers trailing explanations refer to the process step in each respective phase diagram.

4.5.1 Initialization

Any node proposing to be an ambassador eventually broadcasts information to identify themselves and the group they intend to administer. The selection of ambassadors is arbitrary assuming each fulfills expectations to help resolve sybil attack challenges toward members of its group. We assume such cooperation; it could be upheld in practice by verifying identity against some list of trusted “promotable” members. Any member has the potential to be an ambassador. The promoted ambassador generates an identifier and hash chain for the group, then signs and broadcasts the pair <gid, \( L_g^t \)> to all network members. Every user equipped with these values is able to validate anonymous messages and detect sybil attacks from group gid.

4.5.2 Phase I: chain validation

In order to register with a group anonymously, user U must first take preliminary action to protect his identity. By blinding his chain tail, he or she can send it simultaneously with proof of identity without revealing the unique tail value to any recipient.

1. User U randomly generates hash anchor \( L^0 \) and computes the resulting hash chain of length \( t \) as described above in section 2.5(#2). U then obfuscates the hash tail with a blinding
function(\#3) and sends this blinded result, $B_u$, to the ambassador of the group he wishes to join(\#4).

Each user inherently determines the maximum number of messages a chain can accommodate when selecting the chain length, as $t$ establishes the number pseudonyms the chain can provide. If a user ever suspects his chain has been compromised, or is concerned others believe his messages are from a common source, he can generate a new hash chain and register a new tail to replace the current chain that has not been exhausted.

5. Ambassador A decrypts the blinded tail with his private key and uses U’s certificate to verify U possesses the unique key that signed message four(\#6). A then digitally signs $B_u$ with his private key(\#7). He signs $B_u$ based exclusively on knowledge of sender identity; he cannot uncover the content he signs. Encryption is not necessary when returning the signed chain tail(\#8); it cannot be exposed since neither the tail nor the nonce required to unblind the credential is known outside U.

9. Upon receiving and decrypting the signed tail, U unblinds this to reveal his original hash tail now signed with A’s private key(\#10). U now holds credential $C_s$, the equivalent of signed tail: $\{L^1\}_{sk_A}$.

At this point a user is able to anonymously communicate with any other network participant, provided he obtain a signed credential from each intended recipient. However, this simple model requires considerable communication overhead; a credential is needed for one-way communication between every potential sender and receiver. Each participant would need a maximum of $n-1$ credentials in a network with $n$ nodes. Furthermore, the current state does not fulfill all listed objectives. Therefore we add augmentations to create a more complete and secure scheme.

**4.5.3 Phase II: group registration**

Following successful execution of phase I, a user requests membership into the same ambassador’s group. He or she presents ambassador A with the credential A has signed.

12. U sends this signed credential back to the ambassador as a request to register with his
group. In addition, he sends a self-generated symmetric key for encrypting A’s response (#11).

We assume some temporal delay between the validation and registration steps; otherwise an ambassador may deduce a newly accepted member is likely to have run phase I moments earlier. Introducing another participant to sign the tail would not solve this dilemma. Doing so would only shift this deduction ability to the alternate signer since A must query him regarding the signature.

13. The ambassador determines the tail is from a sender he considers legitimate by verifying he previously signed it. Without having discerned the source identity he would not have signed the tail. Ambassador A presents U the encrypted group identifier (signed to verify the source) and the initial hash chain tail, as well as the group member identifier U is assigned (#14) as an act of approved group membership. He or she encrypts this response with the symmetric key U provided. Using a symmetric key preserves group credential confidentiality without revealing
any identifying information. Obviously, providing a user’s private key would compromise his or her anonymity. A user can determine the next valid credential by hashing $L_0^g$, since we assume the most recently expired group key is public knowledge.

Although the ambassador signs only the chain tail, the remaining hash chain values $L^0$ through $L^{t-1}$ are authorized implicitly due to the one-way characteristic of the hash function. By this we mean any link in a chain can be hashed until the result is $L^1$, which proves knowledge of the chain’s anchor, $L^0$. An ambassador must enforce some maximum chain length to prevent continuous hashing. Otherwise a user could provoke a denial-of-service attack by presenting a value outside his or her hash chain.

![Figure 4.3 phase 2: group registration](image)

### 4.5.4 Phase III: message sending

16. U checks it has not exhausted all links in the validated hash chain before sending. When U needs a new chain validated, he sends a request containing $<\text{gid}, L^g_j, n_U, z_U, C_s, \{L^1\}_{pkA}>$ to his ambassador to displace the previous chain tail with a fresh one. The ambassador knows a chain re-validating user is authorized from group credentials and the signed personal credential.

17. U continues by deriving the next unused individual and group links and obfuscating the individual link with a chosen salt value(#18). Every message does not require a fresh salt,
rotation is completely user decision.

19. Whether a broadcast or multicast message, the sender must provide: the next unused link of the group hash chain to prove group membership and his next pseudonym, which is somehow obfuscated with random salt value $s_U$ only the sender knows.

Each individual link need be obfuscated with a key. We call this key *salt* to avoid the connotation of *key* to imply encryption. Obfuscation removes any distinguishable association between users chain values. A recipient can determine if two messages with sequential pseudonyms originated from the same source by deriving $L^j$ from $h(L^{j-1})$. Revealing this relationship violates our message dissociation objective, from some alarming combination of actions could trigger suspicion. Since $h(\{L^{j-1}\}_{sU})$ does not result in $\{L^j\}_{sU}$ this association is eliminated without the need to change salt for every message.

20. The recipient verifies the unencrypted link is of group gid by comparing it to the last received link from a member of that group. He then stores the message digest and pseudonym $\{L^j\}_{sU}$ for future reference(#21).

22. If a recipient receives two messages with the same content he will check if the group pseudonym from each message hashes to the same tail. If so, it confirms both messages were sent from the same group. He then has reason to be suspicious of a sybil attack and proceeds with phase IV. If not, he accepts the message as legit.

4.5.5 Phase IV: sybil speculation

At this point recipient R is suspicious of a duplicate message being sent from a common group. However, he cannot definitely determine whether homogeneous messages are from unique sources in the same group - a false positive. Being concerned with security, users are “guilty until proven innocent” in our protocol. R will reject such a collision by default; U needs some means of quelling this suspicion.

23. Recipient R sends a challenge message to those who sent conflicting messages, declaring each must prove their identity unique to validate the suspicious message. A nonce is used as challenge identification and replay-attack prevention; it is encrypted for challenger identifica-
tion, proving the challenge is from the original intended recipient and not spurious.

25. Now the legitimate users need their respective ambassador to prove their uniqueness to R. Each user sends the challenge description, his group-index, and fresh link $L_j^{g-1}$, where $L_g^j$ was the link used on the challenged message. The transmission includes secret obfuscation salt $s_U$, encrypted for only A to read.

26. The ambassador’s role is to verify a user has used only his single registered user chain. A verifies the salt revealed was actually used in the original message and hashes until verifying $h(L^j) = L^t$. After confirming the user did not attempt to send with pseudonyms from two different chains, he signs the response as endorsement and forwards it to challenger R (#27).

28. The challenging recipient checks the ambassador’s signature and also confirms U submitted a salt value consistent with the original message. Pending this process holds he accepts the challenged message and sends resolve acknowledgment to the appealing user, otherwise he ignores it or takes measures defined by network administrators regarding sybil attacks (#29).

Proving different pseudonyms came from different hash chains confirms the users are unique, but in no way reveals any information regarding their identity with signed credentials. By
confirming the salt consistency his recipient prevents a sender from fooling him by sending a bogus salt in a sybil response. R will determine \( \{L^1\}_{s_U} \) will not match the hash of \( \{L^1\}_{s_U} \) sent in the initial message. This check will also thwart bogus challenge responses.

30. Finally, U generates a new salt value to prevent both A and R from being able to link future pseudonyms back to a similar source. This approach cannot prove a given user attempted a sybil attack when he or she utilizes a different obfuscation key on the duplicate message. By refusing to defend the duplicate message, a receiver cannot prove the user pseudonyms were from the same chain and therefore the same source. So the protocol does not support such non-repudiation in order to uphold absolute anonymity. The nonce prevents a malicious party from replaying messages to frame a challenged participant. Therefore, a party’s unanswered challenge can result in further suspicion. If each user verifies their own message as legit, then an attack was not attempted.

Figure 4.5 phase 4: sybil resolution
CHAPTER 5. FORMAL VERIFICATION - MODEL CHECKING

In this chapter we provide a description of our modeling practices and implementation for the presented protocol. In an effort to formally validate our objectives and claims, we modeled the protocol in Promela (Process Meta Language) for use with the SPIN ModelChecker[(4)]. SPIN was chosen due to Promela’s familiar syntax, SPIN’s acceptance in the research community, and our precious experience using the tool. In the following section, we explain decisions we made in regard to model implementation and abstraction. Section 5.2 describes our processes implementations for small simulations and explain how we expand to include multiple instances of processes. Section 5.3 elaborates on specified properties. We conclude with an analysis of the computational resources required by different models and introduce our intruder model.

5.1 Modeling decisions

We attempt to build the simplest possible model able to uphold claims and reflect actual protocol performance. Cryptographic actions such as encryption and digital signatures are implied rather than literally performed. Such actions exceed our purpose for model checking and add superfluous states. We assume the owner of a given private key is the only one able to read a message encrypted with complementary public key, according to actual practice. Expired pseudonyms are implicitly rejected; we are only concerned with a fresh pseudonym. In the Promela code, a fresh pseudonym contains a subscript ‘nu’ (v), rather than some index j.

The model is split into two sections; protocol phases I and II comprise code segment 1 and phases III and IV comprise code segment 2. Such modulation of code reduces complexity
without affecting the properties we verify. This partition is justified because circumstances do not allow the global properties to be violated in interaction between these segments. We first focus on a simplified model with one instance of each process and then note what modifications were required to expand the model to consider multiple instances of each processes.

5.2 Model description

We now explain selected details of our model source files contained in Appendix A for reference. We do not devote much space to model description, but consider the code to be adequately commented and refer you to [(4)] for any unclear syntax we do not explain. We first explain pervasive syntax and semantics initially and then specify cases specific to each code segment. Last, we explain modifications made to accommodate multiple instances of each process.

5.2.1 Syntax and semantics

Variable names imply content using two notations. A single underscore between previously defined variables represents concatenation of these variables. Similarly, encryption is implied by double-underscore; \(x_y\) represents \(\{x\}_y\).

Channel names are defined as \(cSR\), where \(S\) is the sending process and \(R\) the final recipient process. We use channels to support two-way communication, so this channel is also used when \(R\) replies to \(S\). In segment 1 we define a channel between user \(U\) and ambassador \(A\), in which each message can hold a maximum of six integers.

\[
\text{chan } cUA = \text{[NUM Us]} \text{ of } \{\text{byte, byte, byte, byte, byte, byte}\};
\]

Constant NUM_Us sets the channel capacity. In single-process model, channel capacity is set to zero, indicating rendezvous communication. In rendezvous mode channels can pass messages only through synchronous handshakes between sender and receiver. They cannot store messages[(4)]. A sender must be prepared to send and the recipient waiting for a message in order for a message to be passed.
In every message the first parameter defined is the intended recipient, followed by the return identification as the second parameter. When receiving on a channel, a process checks whether it is the intended recipient using the Promela eval() function. If the first parameter does not match his is value self he will ignore the message in the channel.

\[
cUA \ ? \ eval(self), \ eval(ambU), \ Bs, \ eval(pkX);
\]

We implement a method similar to [(5)] how encrypted values are represented and considered. The adjacent inclusion of pk\textsubscript{X} above implies the preceding value Bs is actually \( \{Bs\}_{pkX} \). The fourth parameter is evaluated to imply only U can read this value as the sole holder of decryption key sk\textsubscript{U}.

### 5.2.2 Code segment 1

Variables Bs, Bu, and Cs are merely values that are considered digitally signed when assigned a value greater than zero and unsigned otherwise. We keep this simplicity since identity and signature verification are subjects of cryptographic practice rather than our protocol.

### 5.2.3 Code segment 2

For segment two we offer explanations for each individual process declared.

#### 5.2.3.1 User

A user can receive two types of messages, either a sybil challenge (SCHAL) or a resolution acceptance (ACPT). After being challenged, the process postpones sending until receiving an acceptance resolution. This practice would prevent congestion caused by any denial-of-service attack and reduces the number of possible states.

#### 5.2.3.2 Receiver

A receiver does not initiate communication, but responds to a sender’s messages with either acceptance or sybil challenge issuance. Integer array \textit{lastKey} stores the last known salt value
from each sender to consider each user increments both sU and cvu.sU upon every sybil challenge. The lastKey structure will be used later for property verification.

5.2.3.3 Ambassador

The entire ambassador loop is *atomic*, a Promela keyword defining the fragment of code is executed indivisibly [(4)]. This is necessary so a given user cannot commit to send between the time his ambassador checks the channel and responds. If a user were to attempt to send between the time his ambassador checks and responds on the channel, both processes could deadlock attempting to send while refusing to listen. We verify this in section 5.3.

5.2.4 Modeling multiple nodes

A few modifications are needed when expanding the model to accommodate multiple instances of each process. Foremost we must provide for adequate communication capacity for multiple processes. The channel functionality shifts from a rendezvous to a queue, now the channel acts as a queue that can store multiple messages among participants. We provide a maximum of U*R slots available, so it is theoretically possible for simultaneous simplex communication between every possible combination of U and R. In reality any U may send multiple messages to a single R so long as not surpassing the capacity of the channel.

U now has multiple recipients to choose from and does so in the following lines:

:: (u_empty==0 && nfull(cUR) && !mail4U)->
sendUR(10, self, M, gidU, lgv, Cv__kU, i, u_empty);
:: (u_empty==0 && nfull(cUR) && !mail4U)->
sendUR(11, self, M, gidU, lgv, Cv__kU, i, u_empty);
:: (u_empty==0 && nfull(cUR) && !mail4U)->
sendUR(12, self, M, gidU, lgv, Cv__kU, i, u_empty);

Although procedurally we interpret this as sender choosing a single recipient, the model
checker will nondeterministically check all possible recipients when running verification. A similar modification is used when multiple ambassadors are instantiated.

Addressing the appropriate recipient and return location also becomes a concern when introducing multiple users. Rather than removing every message from a channel, users and receivers now poll a channel in search of messages with specific qualities.

\[
\text{cUR} \text{ ?} \text{ ?} <\text{eval(self)}, \text{ cv}_-\text{kU}, \text{ m, gidU, nr}, \text{ skR}> \rightarrow \\
\text{cUR} \text{ ?} \text{ ?} \text{ eval(self), cv}_-\text{kU, m, gidU, nr, skR};
\]

If the poll returns true, users remove the first message addressed to themselves from the queue and continues processing. Two question marks instruct it to search the entire queue, rather than only the first message, using the channel as an array structure rather than first in, first out (FIFO) queue.

### 5.3 Property verification

Formal verification allows us to define a property as true or impossible, rather than intuitively claiming it highly likely or unlikely. Most properties are specified and explained using a Linear Temporal Logic (LTL) formula. In the event a property is not verified, SPIN produces an error trail suitable for simulation analysis[(6)]. All properties should hold in both a single and multi-instance process model. A claim held in a single instance model but violated upon expansion indicates either an error in implemented design not apparent until expansion, or an error in the protocol.

We split statements to be verified into two categories with differing purposes. Properties specific to our protocol we term *claims*. Properties that ensure the model code accurately reflects the protocol we term *sanity checks*. In order to validate claims for multiple instances of a process, a claim must be checked for each independent process. We use the universal quantification notation to represent multiple instances of a process. For example:

\[
\forall x \ [pA x \land qAx]
\]
The following properties SPIN considered valid unless we explicitly explain why one failed.

### 5.3.1 LTL syntax

Temporal logic in general is used to determine if some execution path or set of execution paths fulfill a particular property. Properties in SPIN can be specified using a Linear Temporal Logic formula. Our specified properties make use of two LTL temporal operators with axiomatic meaning, EVENTUALLY (<>), IMMEDIATELY (X), and ALWAYS ([])[(6)]. *Eventually* indicates a requirement is true at some points down a traversed path or set of paths. *Immediately* specifies a given state follows a previous one with no possible intervening state between them. *Always* indicates a requirement holds for every path in every state. SPIN translates presented LTL formulas to consider whether the instance *never* holds, requiring us to negate claims we intend to hold positive connotation.

### 5.3.2 Claim verification

These claims are specific to our protocol. We use boolean values, temporal operators, variables, and labels to specify desired claim correctness. In addition, SPIN verifies generic properties of liveliness such as deadlock avoidance, and of safety such as valid end states, neither of which we specifically define here.

**Claim 1**: No recipient can identify a user requesting group membership.

- **knowSndrAx**: Ambassador $x$ can identify the message sender
- **knowCnAx**: Ambassador $x$ can read the provided link credential
- $\forall x: <>(\text{knowCnAx} \& \& \text{knowSndrAx})$

This formula states that an ambassador can never know both a sender’s true identity and hash chain tail simultaneously. Keeping this knowledge disjointed requires an ambassador
cannot distinguish the tail when aware which user sent a message. At some time tail $L^t$ is implicitly subject to our blinding function. When a tail is exposed during group registration, a sender does not disclose itself as the source of the request.

Claim 1 was only tested in code segment 1, because there is no further exposure of a user’s identity at any point after registration. He or she reveals no more identifying information as the process continues, so the scope in verifying this property is sufficient. Similarly, the remaining claims are exclusively checked only in Section 2. These claim deal with holding pseudonym disassociation and sybil detection, neither of which are relevant in segment 1 of our model.

**Claim 2**: No recipient can determine whether independent, obfuscated pseudonyms are from a similar source.

receive$_R$x: Receiver $x$ has received new message
cvEQkey$_R$x: Receiver $x$ received a fresh credential that EQuals a known salt

$\forall x: (<>(\text{receive}_R x \&\& \text{cvEQkey}_R x) )$

This formula states that receivers cannot know a salt value which allows then to determine the original user link. If a link is discovered, it becomes trivial to correlate independent links to a similar hash chain, invalidates our claim. Claim 2 tests the fulfillment of our third objective.

Remember that to quell receiver doubt a user must reveal the secret salt value he used to obfuscate his link. However, once this salt is exposed, the recipient is able to reveal any credential he previously received and stored that was obfuscated with the same value. SPIN verifies this weakness in our model. The salt value obfuscating the user link is assigned to variable $Cv_{-sU}$, which represents a fresh credential obfuscated with salt $sU$. Knowing that respective salt value will reveal $Cv$. If a recipient learns salt value is $s$, he can determine the original user link used in all messages where $Cv_{-sU}$ equaled value $s$ also.

A user cannot resolve a sybil challenge without revealing his or her salt, because an ambassador needs to verify the user sent pseudonyms from his only registered chain. Therefore, users have two options in this situation. The first is proactive; he may anticipatorily change salt prior to obfuscation use to prevent this breach of previous message pseudonyms. He or
she may do this in preparation to solve a sybil challenge and but separate new messages from
previous ones. The other option is reactive. The user may choose a new salt and resend the
challenge message to preserve separation among pseudonyms. Realize the reactive method is
vulnerable to replay to force perpetual sybil suspicion, but replay is detectable and therefore
not beyond preparation. We modify our model to enact the non-resolute option by setting
constant PROTECT_PREV to 1. In this case the challenged user declines to resolve the sybil
challenge and could opt to resend the message to avoid revealing the salt value. Now, not only
does the desired claim hold, but we have another claim that should be noted as well.

**Claim 3**: Claim 2 cannot become false unless a user releases his or her salt value.

\[ releaseUx: \text{User } x \text{ exposes salt } s_U \text{ to a requesting participant} \]
\[ \forall x,y: \neg (releasekUx U (receiveRy \&\& cvEQkuRy)) \]

This formula states that a recipient cannot acquire a salt value to determine previous
messages were sent by a common sender prior to that sender releasing the required salt. The
statement after until (U) is claim 2. Note that if U receives a challenge after he has changed
either his individual or group chain or salt, he has invalidated the integrity of the sybil resolve
process and must resend using the updated credential.

**Claim 4**: All detected sybil challenges are resolved.

\[ chalUx: \text{User } x \text{ cannot receive any future sybil challenges} \]
\[ doneUx: \text{User } x \text{ is in a valid end state} \]
\[ \forall x: (chalUx \&\& doneUx) \]

This formula states that a user never terminates while there exists a possibility of receiv-
ing any sybil challenge. We verify this by ensuring all receivers have terminated, each user
has sent the maximum amount of messages, and no pending sybil challenges reside on any
channel. Although in practice this insight is not possible from a participant perspective, we
accommodate an ideal situation to prove graceful protocol termination.

**Claim 5**: All sybil challenges are detected
Detecting a sybil attack is much more concrete than many other security violations, such as intrusion detection. In our protocol, determination is simply a comparison between two pairs of gid and message content values. Comparing two numbers is a trivial issue not relevant to model checking. Our use of model checking is concerned with how detection may be circumvented with a sequence of actions. SPIN confirmed one issue with sybil attack detection in our protocol.

Consider if a malicious user were to send a message using his validated hash chain and later register a new chain with his ambassador. He could then send a repeat message with a pseudonym from the new chain. The recipient will correctly flag this as an attempted sybil attack and issue a challenge. However, the ambassador eventually will appeal in favor of the malicious user. The ambassador will see no violation because the user has used his sole registered chain for the second message relay. We propose two solutions; both require enforcement by an ambassador. Solution one tasks the ambassador to keep track of previous registered links held by an indexed user. In the second solution ambassadors enforce a critical period in which registered chains cannot be replaced. We chose to enforce the critical period method due to its compatibility with our model. We also believe it is more likely to be implemented to avoid committing additional resources to store and check against multiple previous tails.

\[ \text{acceptAy}: \text{Ambassador } y \text{ accepts a User appeal} \]
\[ \text{ltU}x: \text{Value of User } x's \text{ valid individual hash chain tail} \]
\[ \text{hasSentU}x: \text{User } x \text{ has sent a previous message} \]
\[ \text{crit: Network is in critical period when chain revalidation is not allowed} \]
\[ \langle (\text{acceptAy} \&\& \text{ltU}x \&\& \text{hasSentU}x \&\& \text{crit}) \]}

The formula above checks that an ambassador will never accept a user appeal when a chain was attempted to be registered during a critical period. Adherence to the critical period specification solves his inaccurate judgment to approve iniquitous sybil appeals.
5.3.3 Sanity checks

The claims above are may be invalid if not tested on an accurate representation. Therefore, prior to verifying claims we tested correctness by running “sanity checks”. These checks are generic claims used to verify our model performs as expected. We use them in two ways. First we ensure expected behavior is reached. Second, we purposely modify the model to check cases of expected failure are appropriately reflected as well. Each check is numbered according to its pertaining code segment.

Check 1.1: Ambassadors always authenticate a user’s identity before then signing Bu.

\[ authAx: \text{Ambassador } x \text{ authorizes user } u \]
\[ signsAx: \text{Ambassador } x \text{ signs tail } L^t \text{ of user } u \]
\[ \forall x: (![authAx \rightarrow X signsAx]) \]

This formula states ambassadors always sign tail Bu immediately after authenticating sender U. As refusal to sign a valid tail would render U powerless to proceed any further.

\[ \forall x: [](![signsAx U authAx]) \] This formula states A never signs a credential without first authenticating whomever sent it. Signing a credential without first authenticating would render the use of authentication to be futile.

Check 1.2: Users always confirms credential Cs is signed by an ambassador before registering.

\[ confirmCsUx: \text{User } x \text{ confirms Cs was signed by the expected ambassador} \]
\[ registerUx: \text{User } x \text{ attempts to register with the validating ambassador} \]
\[ \forall x: ![](confirmCsUx \rightarrow <>registerUx) \]

This formula states a user always registers with a group using Cs after it has verified the signer.

\[ \forall x: [](![confirmCsUx U registerUx]) \] This formula states that U never attempts to register
using signed credential Cs before verifying it was signed by the intended ambassador. Obviously an unauthorized user would be rejected should he or she perform disaccordinly.

Check 1.3: All credentials sent from an authenticated user are received in return, signed by their ambassador.

- **sendBuux**: User \( x \) sends blinded link \( Bu \) requesting validation
- **registerUx**: User \( x \) attempts to register

\[
\forall x. [(sendBsUx \rightarrow <> registerUx)]
\]

This formula states every link a user sends to validate will eventually return. In order for users to register they must receive the tail back signed since we restrict them to sending only one unsigned tail. (see check 1.2) Furthermore, an assertion statement ensures the process will error if the credential is returned unsigned.

\[
\forall x. [], ((sendBsUx U registerUx) \rightarrow)
\]

This formula states a user never attempts to register before sending a tail for validation. The implication in the previous claim would be invalid if a user were allowed to register without sending. Upon reception, each user attempts to register. Here we verify that he never registers, and therefore cannot bypass receive, without sending a request.

Check 1.4: Removing the *atomic* declaration from the ambassador procedure causes a deadlock error when less than \(|U|\) slots are available in channel \( cUA \). As expected, removing this restriction allows a user to send between the time his ambassador probes the channel and attempts to send. Furthermore, the atomic quality is necessary to ensure a process does not poll a channel indefinitely.

Check 1.5: Failing to reset the knowledge indicator of either variable \( sndr1 \) or \( cn \) causes the first global claim to evaluate as false. This consequence is expected since such behavior would indicate an ambassador has correlated knowledge of both a user identity and his exposed credential simultaneously.

Check 2.1: Releasing salt always exposes previous messages
releases\textsubscript{Ux}: User \(x\) releases private salt value \(s_\text{U}\)

\[
\forall x, y. \left[\left( \text{releasesUx} \rightarrow \langle\langle \text{receiveRy} \& \& \text{cvEQkuRy} \rangle \right) \right]
\]

This converse of global claim 3 states releasing a salt value will always allow receiver \(y\) to read previous messages. The formula does not hold because releasing salt after a single use would not reveal any prior messages since none were sent. Our model verifies this statement as false.

Check 2.2: Our model includes global variables that provide transparency of inter-processes communication status in order to prove global claim 4. Removing these variables eliminates such universal omniscience and reflects the actual network dilemma when terminating mutual session. We have verified all sybil attacks may not be resolved if the indented user challenged ceases participation before receiving the challenge. Conversely, a recipient can never actually know a sender will never send again.

In addition, we set some of the checks that break the do-loop to always evaluate to false. As expected the loop never stops and results in a violation of liveliness by preventing entrance into a valid end state.

Check 2.3: As in check 1.5, removing \textit{atomic} from blocks of code in user, receiver, or ambassador processes opens the possibility one process commits to send between the time another process has determined the channel suitable for and performing sending. We rightfully are informed of deadlocked states violating liveliness.

Check 2.4: When an ambassador issues a new group hash chain he or she must broadcast this change and send the new anchor to all group members. Similar to the fifth global claim, assume a malicious user were to send a repeat message with a different group credential. In this case the recipient would not even become suspicious of a sybil attack; the group link from the duplicate message would appear to be from a completely different group. We verified our model failed to detect the sybil attack under this circumstance.

Fortunately, a recipient is aware of group chain rotation since all participants must be current of every valid group chain. We propose two options. First, a recipient can invalidate
all votes submitted with the previous chain after that group changes hash chains. Second, it can perform an auxiliary check and compare messages using the new chain to messages sent using the previous chain. This would prevent the need to invalidate past votes. A restricted period cannot be infinite. The restriction must eventually expire some time after the tally is no longer an issue.

Check 2.5: In addition to these, we have multiple assert statements to enforce consistent, expected behavior. Such statements are simple enough they do not require individual explanation. A assert statement will cause SPIN to report an error if violated. Refer to [(4)] for more information regarding the assert keyword.

### 5.3.4 Model checking results

We include the result of checking with SPIN for safety properties only, specified by the `-DSAFETY` option. We limit each user to send two messages not prompted by sybil challenge. The accumulated memory usage is shown rather than a maximum amount used at one instance. We utilized compression by the use of `-DCOLLAPSE`.

<table>
<thead>
<tr>
<th>U,A</th>
<th>States</th>
<th>Transitions</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>10</td>
<td>10</td>
<td>1.57</td>
</tr>
<tr>
<td>2,1</td>
<td>171</td>
<td>222</td>
<td>1.57</td>
</tr>
<tr>
<td>3,1</td>
<td>2,638</td>
<td>4,227</td>
<td>1.78</td>
</tr>
<tr>
<td>4,1</td>
<td>47,113</td>
<td>89278</td>
<td>7.62</td>
</tr>
<tr>
<td>1,2</td>
<td>24</td>
<td>24</td>
<td>1.57</td>
</tr>
<tr>
<td>2,2</td>
<td>558</td>
<td>764</td>
<td>1.57</td>
</tr>
<tr>
<td>3,2</td>
<td>16,812</td>
<td>29176</td>
<td>3.52</td>
</tr>
<tr>
<td>4,2</td>
<td>622,952</td>
<td>1296140</td>
<td>91.28</td>
</tr>
<tr>
<td>1,3</td>
<td>35</td>
<td>35</td>
<td>1.57</td>
</tr>
<tr>
<td>2,3</td>
<td>1,199</td>
<td>1664</td>
<td>1.68</td>
</tr>
<tr>
<td>3,3</td>
<td>52,562</td>
<td>93407</td>
<td>8.64</td>
</tr>
<tr>
<td>4,3</td>
<td>2,867,760</td>
<td>6168410</td>
<td>105.82</td>
</tr>
<tr>
<td>5,3</td>
<td>out of memory</td>
<td>out of memory</td>
<td>out of memory</td>
</tr>
</tbody>
</table>

Increasing process instances and channel slots for added instances specifically cause models to grow quickly. The complexity of models tends to increase exponentially due to the
### Table 5.2  results of model checking segment 2

<table>
<thead>
<tr>
<th>U,R,A</th>
<th>States</th>
<th>Transitions</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1,1</td>
<td>307</td>
<td>435</td>
<td>1.61</td>
</tr>
<tr>
<td>2,1,1</td>
<td>82,309</td>
<td>152,933</td>
<td>13.50</td>
</tr>
<tr>
<td>3,1,1</td>
<td>out of memory</td>
<td>out of memory</td>
<td>out of memory</td>
</tr>
<tr>
<td>1,2,1</td>
<td>3,502</td>
<td>6,651</td>
<td>1.61</td>
</tr>
<tr>
<td>2,2,1</td>
<td>839,967</td>
<td>1,921,360</td>
<td>167.99</td>
</tr>
<tr>
<td>3,2,1</td>
<td>out of memory</td>
<td>out of memory</td>
<td>out of memory</td>
</tr>
<tr>
<td>1,3,1</td>
<td>24,411</td>
<td>57,540</td>
<td>4.69</td>
</tr>
<tr>
<td>2,3,1</td>
<td>6,602,660</td>
<td>17,846,100</td>
<td>1584.64</td>
</tr>
<tr>
<td>3,3,1</td>
<td>out of memory</td>
<td>out of memory</td>
<td>out of memory</td>
</tr>
<tr>
<td>1,1,2</td>
<td>611</td>
<td>867</td>
<td>1.61</td>
</tr>
<tr>
<td>2,1,2</td>
<td>363,415</td>
<td>678,033</td>
<td>65.42</td>
</tr>
<tr>
<td>3,1,2</td>
<td>out of memory</td>
<td>out of memory</td>
<td>out of memory</td>
</tr>
<tr>
<td>1,2,2</td>
<td>7,001</td>
<td>13,299</td>
<td>1.61</td>
</tr>
<tr>
<td>2,2,2</td>
<td>3,454,270</td>
<td>7,900,070</td>
<td>746.12</td>
</tr>
<tr>
<td>3,2,2</td>
<td>out of memory</td>
<td>out of memory</td>
<td>out of memory</td>
</tr>
<tr>
<td>1,3,2</td>
<td>48,879</td>
<td>115,077</td>
<td>11.14</td>
</tr>
<tr>
<td>2,3,2</td>
<td>out of memory</td>
<td>out of memory</td>
<td>out of memory</td>
</tr>
</tbody>
</table>

exponential growth in possible states.

5.3.5  **Validity of claims considering infinite process instances**

The explosion of state space quickly consumes available time and memory resources, and limits how large a model can be simulated using SPIN. We have proven our model to hold claims with a minimum of two instances of process User, two instance of process Receiver and two instances of process Ambassador. Limited resources keep us from infinitely expanding our model to verify claims for limitless number of instances and messages. Therefore, we intuitively assert the properties SPIN has verified for a small set of users also hold when scaled to any number in practical circumstances.

Consider a model with x instances of process X and y instances of process Y and channel with message capacity (x*y). Since the ratio of channel capacity to possible source/destination combinations remain constant, the performance is consistent as the number of process instances increases and state space expands. If transmission from \(X_1\) to \(Y_1\) is possible, then transmission
from $Y_1$ to $X_1$ is possible as well. For every possible combination there is two-way communication possible, inferring the required communication bandwidth is available. If $X_1$ to $Y_1$ communication is not possible, then the protocol could not be initiated; but protocol initialization failure cannot contradict our claims under these circumstances.

Order of events would also be a concern of expansion in procedural languages. However, since Promela defines that SPIN checks all possible states non-deterministically, sequence is never an issue because all cases are always considered. In conclusion, we hold no doubt that any claim that holds with a small number of process instances will also hold for any number.
CHAPTER 6. ANALYSIS AND DISCUSSION

This chapter contains a discussion of observations that were not checked using any formal method. Some transcend our experience with model checking tools, others may be able to be checked in future work, and still others are more effectively assessed with a method besides model checking. We highlight and discuss some insights we consider interesting. Section two does more of the same in regard to security. Section three considers protocol maintenance.

6.1 Informal observations

6.1.1 Group size assessment

Though there is little correlation balance between group size and anonymity, we intuitively determine that smaller groups are more advantageous. Only in regard to the group ambassador does group size affect anonymity. An ambassador knows the true identity of all participants who could possibly be a current group member. He can identify messages sent by users he has validated based on the included group credential but knows nothing more. Furthermore, the only time an ambassador can know for certain a member is active in his group is when the number of tails he has signed equals the number of registered group members. Otherwise there is some uncertainty as to who the inactives parties are. Apparent randomness of user pseudonyms prevents any external observer from keeping count of individual senders per group. He may only attempt to gain insight by how frequently messages from a group are transmitted.

On the other hand, when considering hash collision by relaxing assumption five, the probability of false positive sybil speculation increases with group size and coincidental hash conflict. In addition, maintenance overhead increases with group size, specifically when a member leaves
their group. We discuss this maintenance more in section 6.3.

### 6.1.2 Storage requirements

Each participant would be required to consistently store a considerable log of reference information during authorization and message repeats. Every participant must store a valid chain of pseudonyms and index values for both their group and individual. Each must store the most recently expired link from each group and message digest, gid pair of all previous messages. We envision a high-efficiency storage method could be used, such as a bloom filter\[21\]. A bloom filter is convenient and suitable since we are interested in identifying a repeated message rather than message contents. An ambassador must store a bit more in addition to generic user requirements. He must always know his group identifiers but more importantly must keep track of each group member’s validated hash tail, which is indexed according to the identifier assigned at registration. In programming terms, this can be thought of as a two-dimensional array: $[z_{U_i}] [L_{U_i}^t]$.

### 6.1.3 Computation requirements

Each participant must consider every message he or she observes and store the related credential to prevent accepting outdated pseudonyms. All must compute the hash of every message for sybil detection. Although naively accepting a group credential is valid and would save a drastic amount of computation, it is subject to accepting tandem spoofed links with a valid group identifier. In anticipation of a sybil attack, a user must compute and compare to stored message digests for matching $<$gid, obfuscated link$>$ pairs.

### 6.1.4 Communication requirements

To fulfill the computation requirement, every node must actively monitor each message on a network. Many network interface devices are able to function in a “promiscuous” mode where it considers all traffic instead of ignoring messages once it has determined it was not the intended recipient. Network organization and message sending take little consideration compared to
current network load. However, resolving sybil challenges requires four transmission among three nodes, meaning it is probably more desirable to simply resend when some limitation is not in place. Users constantly leaving a group would also cause increased communication. The ambassador would broadcast a new group chain tail, then issue new credentials to all remaining group members.

6.2 Security

In this section we analyze possible attacks and the ability of our scheme to detect, prevent, and respond to each. We consider and note a member in the same group as their target provides some advantage for specific attacks.

6.2.1 Sybil attacks

The use of pseudonyms raises the ability to enact multiple identities. We implement a multi-credentialed model that allows detection and prevention of a sybil attack, and allow a benign user to vindicate himself without compromising anonymity. We do not consider it any further here since we formally verify the effectiveness of our scheme in detecting sybil attacks with model checking.

6.2.2 Replay attacks

We require our second assumption in multi-sender or multi-receiver cases to prevent replaying expired group links. Many protocols, such as Kerberos\cite{19}, include a tamper-resistant time stamp signed by a private key of the sender. Including a private key would introduce identifying information into a message, contradicting our effort to ensure anonymity. Several of our steps may be potentially vulnerable with an unsigned nonce. Some replay attack may allow unconsidered sybil attacks. In the future, we hope to formally verify resistance to replay attacks with model checking.

The expired pseudonym issue easily handled if an ambassador acted as a group communication hub. The ambassador could easily manage group credentials centrally if all outgoing
messages were required to junction through him. However, we wish to avoid such reliance out of concern for communication bottlenecks, denial-of-service from attacks or malfunction.

6.2.3 Denial-of-service attacks

Two specific actions are somewhat accommodating to DoS attacks. The computational requirements of public-key cryptography make any unnecessary increase in encryption or decryption a considerable resource drain. A malicious user could replay unencrypted messages for denial-of-service attacks. Any time an attacker wants to invalidate a user’s message, he replicates the message and sends it with his own credential chain. This requires the attacker know the group credential, which is trivial if a valid member of the same group. The challenged target initiates the taxing sybil resolution process. Ultimately the challenger rejects the extra message due to unresolved conflict because attacker declines the recipient’s sybil challenge. The attacker merely sought to cause the resolution process.

6.2.4 Loquacious ambassador

Although the ambassador is needed to resolve sybil attack speculation and has stored credentials, he is never able to use the information he knows for malicious purposes. He knows user chain tails, but the one-way quality of hash functions prevents any abuse; these chains are useful only for verification. Similarly he cannot initiate separate group identifier, credential pairs because each is bound to his identity by public key.

6.3 Maintenance

Our protocol does not restrict the dynamic nature of an ad hoc network. However, there is considerable maintenance to consider in such a specialized environment. A new member is simply able to register with no further overhead of group credentials. Unfortunately, a new group chain must be issued to the remaining members in the eventa participant leaves the group. This prevents relinquished members from abusing knowledge of multiple group
credentials. We assume a user informs the ambassador of his departure, or some level of inter-ambassador check is performed.

Interrupted service from an acting ambassador due to malfunction or hijack would be somewhat disruptive, but not irrecoverable. Replacement of a defunct or lost ambassador essentially entails no more overhead than the initialization process. An ambitious group member can issue group credentials and broadcast itself as replacement ambassador for group gid, current group members will be accepted by all other network members.
CHAPTER 7. CONCLUSION AND FUTURE WORK

In this paper we have presented a protocol for providing authenticated, authorized communication while upholding anonymity. The protocol effectively detects and prevents sybil attacks, and provides non-linkable messages as well. Our contributions are confirmed by fulfillment of objectives:

1. In step four of phase 1, the act of sending the blinded hash tail in combination with proof of identity to an authority fulfills this objective.

2. Abstracting user identity by use of group credentials, with support of the blind signature exchange allows a user to prove his authorization without revealing the credential.

3. Obfuscation of user hash links removes the ability to derive other links with a hash. We verify this in global claim three.

4. In section 5.3 we formally verify the aptitude of our protocol to detect and withstand sybil attacks, as well as resolve all false positives, that do not replay exterior credentials.

5. We have explained our use of SPIN and presented the results of formally verifying claims.

6. We discuss the capabilities of our protocol to accommodate a changing physical environment of an ad hoc network in section 6.3.

Despite having propitious qualities, our protocol has some limitations and challenging requirements needing consideration in future work. Most troubling is the assumption of synchronized group credentials, since acting as enforcement it is not in the interest of neighboring nodes. Devising an effective method to detect sybil attacks without group formation may also remove considerable overhead. The most limiting restriction of our protocol may not be the practicality, but rather the toll on resources and time. In anticipation of use in sensor networks we seek to make the protocol less taxing on resources. Sensor nodes are currently un-
able to accommodate the computationally expensive practice of public key infrastructure. An anonymous key pair may provide an alternative means\cite{(20)}. Capacities for storing previous messages is also a concern in sensors.

If a recipient respectively receives $L^4$, $L^2$, $L^3$, then $L^3$ will be rejected because all previous pseudonyms were expired by his witness to lower link $L^2$. \cite{(3)} presents a solution for receiving disordered credentials that may remedy the issue in our protocol as well. We seek a deeper analysis of how collusion may impact the protocol, either among multiple malicious participants, or a traitor who exposes credentials meant to be kept private. We wish to analyze this formally with model checking, in addition to checking replay attacks as mentioned in section 6.2.2.
APPENDIX A. SOURCE CODE

Promala source code; both for two users and two recipients.

Code segment 1

/*
 * Validate & Register - segment1.pml
 * This file combines Phase I and Phase II into a single execution model. This decision is
 * valid since the overarching global claims only pertain to these two phases and cannot
 * be violated on exited.
 */

#define NUM_As 2 /* number of Amabssador processes instances */
#define NUM_Us 2 /* number of User processes instances */

byte pkA, pkU, Cgw, nix; /* nix is a null value of no significance*/
chan cUA = [NUM_Us] of {byte, byte, byte, byte, byte, byte};

/*
 * User - generic node
 */
proctype User (byte self, prikU, nU, kU) /* #1 (nU), #12(kU) */
{
 byte ambU, Cn; /* values generated */
byte skU, gidU, zU, cg0, cgw; /* values received */
byte Cs, Bu, Bs;
byte Lt; /* #2*/

if /* choose a group ambassador */
::ambU=20;
::ambU=21;
::ambU=22;
fi;

byte pk_ambU = ambU*10+3; /* retrieve ambassador's public key */

sendBu:
cUA ! ambU, self, Bu, pkA, nix, nix; /* #4: Send [dest], {Uid, CertU, Bu}PubkA */

cUA ?? eval(self), eval(ambU), Bs, _, _, _;

confirmCs:
assert(Bs); /* # ensure Bs is signed (Bs==1)*/
Cs = Bs; /* #10 verify Cs = {Cn}skA */

printf("have signed credential \%d\n",Cs);

register:
cUA ! ambU, nU, Cs, Cn, kU, pkA; /* #12 Send to A: [Aid], {Cn, {Cn}PrikU, kU}PubkA */

printf("sent\n");
cUA ?? eval(nU), gidU, zU, cg0, cgw, eval(kU); /*#15: eval(kU) implies decryption */
printf("got group credentials\n");
}
Ambassador - group authority

proctype Amb(byte self, prikA, pubkA) {
  byte gid, Cg0; /* values generated */
  byte sndr1, nu, cn; /* values received */
  byte ku, zu, cs, bu; /* values generated */

  end:
    do
      ::
      atomic
      {
        cUA ?? <eval(self), sndr1, bu, eval(pkA)> ->
        if
          :: cUA ?? eval(self), sndr1, eval(0), eval(pkA), _, _; /* #5 can decrypt with skA*/
        printf("signature requested");
        /* A learns Bu and certU*/

        authenticate: /* #6 verifies U with CertU */
        sign:
          bu=1; /* #7 A signs Bu to produce Bs */
          printf("signed\n");
          cUA ! sndr1, self, bu, nix, nix, nix; /* #8: send [uid, Aid, null],[Bs]pkU to U */
          sndr1=0;

          :: cUA ?? eval(self), nu, eval(1), cn, ku, eval(pkA);
          /* #13 eval(1) verifies: Cn=\{Cn\}PriKa. Proves Cs was signed by self. */
          register:
          printf("membership request accepted");
      }
cUA ! nu, gid, zu, Cg0, Cgw, ku; /* #14: send to nu_owner: nU, {gid, zu, Cg0, Cgw}kU*/
fi;
cn=0;
}
}

/*
 * initiate - missions control
 */
init
{

atomic
{
    run User(01, 014, 015, 016);
run Amb(20, 204, 203);

    run User(02, 024, 025, 026);
    run Amb(21, 214, 213);
}
}

/*
prefixes:
Users:0
Recs: 1
Ambassadors:2

suffixes
pk:3
Code segment 2

/*
* segment2_test.pml
* For testing protocol specific claims with claims/_.ltl files.
* Segment 2 models the behavior of phases I and II.
* #x corresponds to the step number in the diagram
*/

#define NUM_Us 2 /* number of U process instances to run */
#define NUM_Rs 2 /* number of R process instances to run */
#define PROTECT_PREV 0 /* boolean to determine U's behavior in releasing kU */
#define MAX_MESSAGESU 2 /* max # of messages each U can send */

/*@ constant to check if a process instance has a message waiting in channel UR */
#define mail4U cUR ?? [eval(self), Cv__kU, m, gidU, nr, skR]

byte regTail[5]; /* current registered tail of each user */
byte lastKey[5]; /* most recent expired key of each user */
byte gidu[5]; /* group id a respective member belongs to */

mtype = {M, /* generic message content */
APEAL, /* appeal to ambassador */
CHNINIT, /* initiate new user hash chain */
ACPT, /* sybil appeal acceptence message */}
SCHAL, /* sybil challenge message */
SYBL}; /* indication of sybil attack */

byte pkA, pkU, pkR, skA, skR, gid_Cgj, nix;

byte sybil_chal; /* quantity of unresolved sybil challenges active */
byte u_done=0; /* quantity of terminated user instances */
byte r_term=0; /* quantity of terminated receiver processes */

bit newChain=0; /* indicates action of reregistered chain,
* only allow one chain, reduces state space */
bit crit=0; /* critical section indicate */

/* bidirectional communication channels between processes. The capacity of
* channel cUR must be manually modified; it should accommodate (NUM_Us*NUM_Rs)
*/
chan cUA = [NUM_Us] of { byte, byte, mtype, byte, byte, byte, byte, byte };
chan cAR = [NUM_Rs] of { byte, byte, byte, byte, byte, byte, byte, byte, byte };
chan cUR = [2] of { byte, byte, mtype, byte, byte, byte, byte, byte, byte, byte };

/*
* "method to send messages on channel cUR
* input: recurrence, self:sender, m: message, gidu:sender gid, lgv:new group link
* cv__ku: obfuscated user credential, j:sender message counter, ue:senders u_empty variable
*/
inline sendUR(recv, self, m, gidu, lgv, cv__ku, j, ue)
{
  cUR ! recv, self, m, gidu, lgv, cv__ku, j, ue;
  j--; /* decrement number of messages left to send */
/* user checks if it has exhausted the number of messages allowed */
if
:: (j<=0) -> ue=1; u_done++; 
:: else
fi;
}

/*
* User - generic node who initiates communication
*/
proctype User (byte self, prikU, nU)
{

byte sc=0; /* indicates if is resolving sybil challenge*/
bit u_empty=0; /* whether U has reached the maximum sending limit */
byte zU, gidU; /* assigned values from A */
byte kU, ambU, Cv, Ch, lgv, Cv__kU,i, LtU; /* values generated or derived from knowledge */
byte nr, cv__kU; /* values received from channel */
mtype m; /* message holding value */

/* choose available ambassador */
if
:: ambU=20;
/*::: ambU=21; */
fi;

zU = self;
kU=self;
Cv__kU = self;
gidU = ambU;
i=MAX_MESSAGESU;
do
::
atomic{

if
	/* poll channel for mail addresses to user */
	:: cUR ?? [eval(self), cv__kU, m, gidU, nr, skR] ->

cUR ?? eval(self), cv__kU, m, gidU, nr, skR;

if
::(m==SCHAL)->
if
:: !PROTECT_PREV ->
sc++;
releasekU:
cUA ! ambU, zU, APEAL, Cv__kU, nr, gidU, kU, LtU;

cUR ?? eval(self), cv__kU, eval(ACPT), gidU, eval(nr), skR;
sybil_chal--;
sc--;
kU++; Cv__kU++;

/* enable linkability validation. In this case U declines to respond to a
sybil challenge to protect message dissociation */
:: PROTECT_PREV ->
sybil_chal--; /* decrement gobal chellange counter */
sc--; /* reduce count of outstanding challenges*/
kU++; Cv__kU++; /* symbolically change key*/
fi;
fi;

/* #19. choose which recipient to send to. Note a listing need to be manually added or removed
* for each instance Receiver process.

*/

:: (u_empty==0 && nfull(cUR) && !mail4U) -
sendUR(10, self, M, gidU, lgv, Cv__kU, i, u_empty);
:: (u_empty==0 && nfull(cUR) && !mail4U) -
sendUR(11, self, M, gidU, lgv, Cv__kU, i, u_empty);
/* :: (u_empty==0 && nfull(cUR) && !mail4U) -
sendUR(12, self, M, gidU, lgv, Cv__kU, i, u_empty);
*/

/* attempt to send repeated message to recipient with id 10. He should detect this
* to be a sybil attack.
*/
:: (u_empty==0 && nfull(cUR) && sc<=1 && sybil_chal<NUM_Us-1) -
sendUR(10, self, SYBL, gidU, lgv, Cv__kU, i, u_empty);

/* #16. initialize new chain with ambassador*/
:: (u_empty==0 && nfull(cUA) && !newChain && !crit && !mail4U) -
LtU++;
cUA ! ambU, zU, CHNINIT, Cv__kU, 0, gidU, LtU, pkA;
cUA ?? eval(zU), ambU, m, cv__kU, _, gidU, _, _;
i--;
if
:: (i<=0) -> u_empty=1; newchain:u_done++;
:: else
fi;
newChain=1; /* indicate a new user chain has been registered */

/* group chain changes*/
/*
:: if
:::(gidU != 20) -> gidU=20;
::(gidU != 21) -> gidU=21;
fi;
 */
fi;
}

od unless { u_empty && sc==0 && r_term>=NUM_Rs};

/* valid end state with sanity checks
* entered when no messages are left to send, pending cybil challenges are resolved, and
* all recievers are terminated to ensur eno further challenges can be issued.
*/
end:
assert(u_empty);
assert (u_done >=NUM_Us);
assert(empty(cUR));
assert(empty(cUA));
assert(empty(cAR));
assert (i ==0);
printf("U DONE %d\n",self);
}
/* end proctype User */

/*
* Reveiver - message recipient
*/
proctype Receiver(byte self, prikR, nR)
{
 byte gid, lgv, ku, cvu_kU, sndrU, sndrA, valNr;
mtype mv; /* stores new message */

do
::
atomic{
if
:: cUR ?? [eval(self),sndrU, mv, gidu, lgv, cvu__kU] ->
cUR ?? eval(self), sndrU, mv, gidu, lgv, cvu__kU; /* #5 can decrypt with skA*/

lastKey[sndrU]=cvu__kU; /* store last expired user pseudonym */
receive:
if
/* sender attempted sybil attack */
:: (mv==SYBL) ->
if
::assert(gidu[sndrU]==0 || gidu[sndrU]==gidu) ->
sybil_chal++;
cUR ! sndrU, cvu__kU, SCHAL, gidu, nR, prikR; /* #23. send sybil challenge */

::(gidu[sndrU]!=gidu) ->
printf("Breach");
fi;
:: (mv!=SYBL) ->
printf("OK");
fi;

gidu[sndrU]=gidu; /* store group of sender */

/* poll for appeal message from ambassadors */
:: nfull(cUR) && cAR ?? [eval(nR), sndrA, gidu, lgv, cvu__kU, ku, skA, pkR] ->
cAR ?? eval(nR), sndrA, gid, lgv, cvu__kU, ku, skA, pkR, sndrU;
cUR ! sndrU, cvu__kU, ACPT, gid /* removed */, nR, prikR; /* #29. send appeal acceptance to user */
/* non-deterministically decide whether to enact a critical-period */
if
::crit -> crit=0; /* if in critical period, end restriction*/
::crit=1
fi;
fi;

}
end:

r_term++;
printf("R DONE");

assert (sybil_chal==0);
assert (u_done>=NUM_Us);
assert (empty(cUR));

*/
/* Ambassador - group authority */
* proctype Amb(byte self, prikA)
  {
}
byte n1, lgv, gid, k1, cvu__kU, zu, ltu, ltv;
mtype m;
end:
do::
    atomic{
        if /* poll for fresh chain registration*/
            ::cUA ?? [eval(self), zu, eval(CHNINIT), cvu__kU, n1, gid, ltv, eval(pkA)]->
            cUA ?? eval(self), zu, eval(CHNINIT), cvu__kU, n1, gid, ltv, eval(pkA);
            regTail[zu] = ltv;
            cUA ! zu, self, M, cvu__kU, 1, gid, ltv, pkA;
        }/* poll for sybil challenge appeal*/
        :: cUA ?? <eval(self), zu, eval(APEAL), cvu__kU, n1, gid, lgv, ltu>--
        cUA ?? eval(self), zu, eval(APEAL), cvu__kU, n1, gid, lgv, ltu;/* #25 check [i][Cn] */
        assert(regTail[zu] == ltu); /* #26 ensure currnt tail is accurate*/
        acptPlea:
            cAR ! n1, self, gid, lgv, cvu__kU, k1, prikA, pkR, zu; /* #27 refer U’s plea to R */
    }fi;
}od;

/*@ initiate - misison control */
init{
sybil_chal=0;

atomic
{
    run User(01, 014, 015);
    run Receiver(10, skR, 105);

    run User(02, 024, 025);
    run Receiver(11, skR, 115);

    run Amb(20, 204);
}

} */

prefixes:
Users: 0
Recs: 1
Ambassadors: 2

suffixes:
pks: 3
sks: 4
nonce: 5

*/
BIBLIOGRAPHY


