Formalizing Provable Anonymity in Isabelle/HOL

Yongjian Li[*,a], Jun Pang[b]

[a] *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China*
[b] *Computer Science and Communications, Faculty of Science, Technology and Communication, University of Luxembourg, Luxembourg*

Abstract

We formalize in a theorem prover the notion of provable anonymity proposed by Garcia et al. [11]. Our formalization relies on inductive definitions of message distinguish ability and observational equivalence on traces observed by the intruder. Our theory differs from its original proposal and essentially boils down to the inductive definition of message distinguish ability with respect to a knowledge set. We build our theory in Isabelle/HOL to have a mechanical framework for the analysis of anonymity protocols. Its feasibility is illustrated through case studies of the Crowds and Onion Routing protocols.

1. Introduction

With the rapid growth of the Internet community and the rapid advances in technology over the past decades, people are getting used to carry out their daily activities through networked distributed systems providing electronic services to users. In these systems, people become more and more concerned about their privacy and how their personal information have been used. Typically, anonymity is a desired property of such systems, referring to the ability of a user to own some data or take some actions without being tracked down. This property is essential in systems that involve sensitive personal data, like electronic auctions, voting, anonymous broadcasts, file-sharing and etc. For example, users want to keep anonymous when they visit a particular website or post their political opinions on a public bulletin board.

Due to its subtle nature, anonymity has been the subject of many theoretical studies [27, 14, 13, 2, 11, 23, 25] and formal verification. The proposed definitions aim to capture different aspects of anonymity (either possibilistic [27, 14, 11, 23] or probabilistic [13, 2, 7]), and formal verification

*Corresponding author
Email addresses: lyj238@ios.ac.cn (Yongjian Li), jun.pang@uni.lu (Jun Pang)

[1] The first author is supported by grants (No. 60833001, 60496321, 60421001) from National Natural Science Foundation of China.

treats systems in different application domains, such as electronic voting systems [19, 6, 16], electronic cash protocols [21], file sharing [30, 4], and electronic healthcare [10]. However, automatic approaches to the formal verification of anonymity have mostly focused on the model checking approach on systems with fixed configurations [27, 28, 5, 7], while theorem proving is a more suitable approach when dealing with general systems of infinite state spaces [17]. We address this situation by investigating the possibility of using a powerful general-purpose theorem prover, Isabelle/HOL [22], to semi-automatically verify anonymity properties.

The idea of expressing anonymity properties in epistemic logic is widely used in the area of anonymity and information hiding [11, 15, 1]. The notion of observational equivalence of traces plays an important role in the epistemic framework of provable anonymity [11]. Two traces are considered equivalent if an intruder cannot distinguish them, i.e., he cannot find any meaningful difference. The distinguishing ability of the intruder is formalized as the ability to distinguish two messages, which is in turn based on message structures and relations between random looking messages. Central in the framework proposed by Garcia et al. [11] is the *reinterpretation function*. Proving two traces equivalent essentially boils down to the existence of such a reinterpretation function. Within their framework, Garcia et al. also define epistemic operators and use them to express information hiding properties like *sender anonymity* and *unlinkability*.

Observation equivalence is basically assumed, whereas such a relation is constructed manually in [11]. However, so far no one has formalized observation equivalence and the construction of this relation mechanically. These are the main obstacles to formalize the anonymity theory. In this paper, we fill the gap by formalizing provable anonymity [11] in the theorem prover Isabelle/HOL.

Our contribution. The main contribution of this paper is twofold: an inductive theory of provable anonymity and its formalization in a theorem prover. We briefly discuss the novelties of our work below:

- We introduce an inductive definition of message distinguishability, which is believed to be a fundamental concept. More precisely, the intruder can uniquely identify any plain-text message. Furthermore, the intruder can distinguish any encrypted message for which he has the decryption key, or which he can construct himself. The observational equivalence between two messages, which can be then lifted to traces inductively, is naturally defined as the negation of message distinguishability. Namely, two messages are observationally equivalent for an agent if he cannot distinguish them according to his own knowledge.

- We propose the notion of *alignment* between two message sequences. Intuitively, alignment requires that the relation, composed of the corresponding message pairs of the two message sequences, should be *single_valued*. Furthermore, the single_valued requirement should remain valid after applying the analyzing and synthesizing operations pairwisely to the message

pairs in the relation. Combining the alignment requirement with the observational equivalence between two messages, we propose an (adapted) definition of observational equivalence between two traces. Thus, our framework can naturally incorporate the concept of reinterpretation function which is extensively used in [11].

- We proceed to formalize anonymity properties in an epistemic logic framework as in [11]. Box and diamond operators are formalized at first, then sender anonymity and unlinkability are defined accordingly.
- We inductively define the semantics of an anonymity protocol, e.g., Onion Routing, as a set of traces, and the relaying mechanism of the protocol is formally defined as a set of inductive rules. Furthermore, we formally prove that the protocol realizes anonymity properties such as sender anonymity and unlinkability under some circumstance by providing a method to construct an observationally equivalent trace for a given trace. We believe that this construction method is generally applicable.
- We build our theory in Isabelle/HOL [22] to have a mechanical framework for the analysis of anonymity protocols. We illustrate the feasibility of the mechanical framework through cases studies on Crowds [26] and Onion Routing [12, 29].

Presentation of the paper. In this paper, we assume readers have some knowledge with Isabelle/HOL syntax. We give a brief introduction to some Isabelle concepts, notations and commands in the appendix. Therefore, we present our formalization directly without elaborated explanation. Notably, a function in Isabelle/HOL syntax is usually defined in a curried form instead of a tuple form, that is, we often use the notation $f\ x\ y$ to stand for $f(x, y)$. We also use the notation $[\![A_1; A_2; ...; A_n]\!] \Longrightarrow B$ to mean that with assumptions $A_1, \ldots, A_n$, we can derive a conclusion B. Here, we briefly introduce some functions on lists, which will be used in later sections of the paper: $x\#xs$ for the list that extends xs by adding x to the front of xs, $[x_1, ..x_n]$ for a list $x_1\#..x_n\#[]$, $xs@ys$ for the result list by concatenating xs with ys, $xs!i$ for the i^{th} element of the list xs (counting from 0 as the first element), $\mathsf{set}\ xs$ for the set of all the elements in xs, $\mathsf{length}\ xs$ for the length of the list xs, $\mathsf{last}\ xs$ for the last element of the list xs, $\mathsf{zip}\ xs\ ys$ for the functions which zips two lists xs and ys to generate a list of pairs, and $\mathsf{map}\ f\ xs$ for the function which applies f to each element in xs. More information on our choices of notations can be found in the appendix.

Structure of the paper. Sect. 2 provides a preliminary introduction to notations and terminologies. Distinguishability and observational equivalence of messages are formally defined in Sect. 3. Then we introduce the notion of alignment for two sequences of messages in Sect. 4. Observational equivalence of traces is formally defined in Sect. 5. Epistemic operators and formalization of anonymity properties are presented in Sect. 6. We model and analyze Crowds and Onion Routing in Sect 7 and Sect. 8, respectively. We conclude the paper with some future research topics in Sect. 9.

This article is a revised and extended version of [20] that appears in the proceedings of the 6th International Conference on Conference on Availability, Reliability and Security. In this version we have included a new notion of alignment, which is crucial for the defintion of observational equivalence and leads to a revised formalization of provable anonymity in Isabelle/HOL. We have extended the case study on Onion Routing accordingly, and conducted a new case study on Crowds.

2. Preliminaries

2.1. Agents, messages and events

Agents send or receive messages. There are three kinds of agents: the server, the honest agents, and the spy. Formally the type of agent is defined as follows:

$$\mathsf{agent} \quad ::= \quad \mathsf{Server} \quad | \quad \mathsf{Friend}\ N \quad | \quad \mathsf{Spy}$$

We use bad to denote the set of intruders, which at least includes the agent Spy. If an agent A is not in bad, then A is honest.

The set of messages is defined using the following BNF notation:

$$\begin{array}{rcl} h & ::= & \mathsf{Agent}\ A \quad | \quad \mathsf{Nonce}\ N \quad | \quad \mathsf{Key}\ k \quad | \\ & & \mathsf{MPair}\ h_1\ h_2 \quad | \quad \mathsf{Crypt}\ k\ h \end{array}$$

where A is an element from agents, N and k from natural. Here, we use k^{-1} to denote the inverse key of k for brevity. MPair h_1 h_2 is called a composed message. Crypt k h represents the encryption of message h with k.

In an asymmetric key protocol model, an agent A has a public key pubK A, which is known to all agents, and a private key priK A. pubK A is the inverse key of priK A, and vice versa. In a symmetric key model, each agent A has a long-term symmetric key shrK A. The inverse key of shrK A is itself. We also assume that (1) asymmetric keys and symmetry keys are disjoint; (2) the functions shrK, pubK and priK are injective, e.g., if shrK A = shrK A' then $A = A'$. In the following, we abbreviate Crypt k h as $\{\!|h|\!\}_k$, and MPair $h_1 \ldots$ MPair h_{n-1} h_n as $\{\!|h_1, \ldots, h_{n-1}, h_n|\!\}$. Such abbreviations are supported in Isabelle/HOL by syntax translation [22].

Operators parts, analz, and synth are inductively defined on a message set H. Their definitions are taken from [24] and tailored for our purposes. Usually, H contains a penetrator's initial knowledge and all messages sent by regular agents. The set parts H is obtained from H by repeatedly adding the components of compound messages and the bodies of encrypted messages. Formally, parts H is the least set including H and closed under projection and decryption.

```
inductive_set parts::  "msg set⇒msg set"
      for H::  "msg set" where
          Inj [intro]:  "x∈ H ⇒ x∈ parts H"
         |Fst:  "{|x,y|} ∈ parts H ⇒ x∈ parts H"
         |Snd:  "{|x,y|} ∈ parts H ⇒ y∈ parts H"
         |Body:  "Crypt k x∈ parts H ⇒ x∈ parts H"
```

The parts operator can be used to define the subterm relation $\sqsubset$: $h_1 \sqsubset h_2 \equiv h_1 \in \mathsf{parts}\{h_2\}$. Note that k is not considered as occurring in $\{\!|g|\!\}_k$ unless k is a part of g.

Similarly, analz H is defined to be the least set including H and closed under projection and decryption by known keys. Note that we use `invKey k` to formally denote the inverse key of `Key k` in our formalization.

```
inductive_set analz::  "msg set⇒msg set"
      for H ::  "msg set" where
          Inj [intro,simp] :  "x∈ H ⇒ x∈ analz H"
         |Fst:  "{|x,y|} ∈ analz H ⇒ x∈ analz H"
         |Snd:  "{|x,y|} ∈ analz H ⇒ y∈ analz H"
         |Decrypt [dest]:"[|Crypt k x∈ analz H;
            Key(invKey k)∈analz H|]⟹x∈ analz H"
```

The set synth H models the messages a spy could build up from elements of H by repeatedly adding agent names, forming compound messages and encrypting with keys contained in H. synth H is defined to be the least set that includes H, agents, and is closed under pairing and encryption.

```
inductive_set synth::  "msg set⇒msg set"
      for H ::  "msg set" where
          Inj [intro,simp] :  "x∈ H ⇒ x∈ synth H"
         |Fst:  "[|x|} ∈ synth H,y|} ∈ synth H ⇒ {|x,y|} ∈ synth H"
         |Encrypt [dest]:"[|k∈ synth H;
            x∈synth H|]⟹Crypt k x∈ synth H"
```

A protocol's behavior is specified as the set of possible traces of events. A trace model is concrete and easy to explain. A trace is a sequence of events. An event is of the form: Says A B m, which means that A sends B the message m. For an event ev = Says A B m, we define msgPart $ev \equiv m$, sender $ev \equiv A$, receiver $ev \equiv B$ to represent the message, sender and receiver of ev. Function initState A specifies agent A's initial knowledge. Typically, an agent's initial knowledge consists of its private key and the public keys of all agents.

The function knows A tr describes the set of messages which A can observe from the trace tr in addition to his initial knowledge. Formally,

```
knows A []= initState A
knows A ((Says A' B m)#evs)=
         if (A=Spy)∨ (A'=A) ∨ (A=B)
         then {m} ∪ knows A evs
         else knows A evs
```

The set used evs formalizes the notion of freshness. The set includes the set of the parts of the messages sent in the network as well as all messages held initially by any agent.

```
used []= ⋃ B. parts (initState B)
used ((Says A B m)#evs)= parts{m} ∪ used evs
```

Function noncesOf $msg \equiv \{m.\exists n.m \sqsubset msg \wedge m = \mathsf{Nonce}\ n\}$ defines the set of nonces occurring in the message msg. The formula originates $A\ m\ tr$, means that A originates a fresh message m in the trace tr. Formally,

```
originates A m []= False
originates A m ((Says A' B' msg)#evs=
         if (originates A m evs)
         then True
         else if (m⊏ msg ∧ A=A') then True
             else False
```

The predicate sends $A\ m\ tr$ means that A sends a message m in an event of the trace tr. Formally,

```
sends A m []= False
sends A m ((Says A' B' msg)#evs)=
            if (m⊏msg ∧ A=A') then True
            else sends A m evs
```

The predicate regularOrig $m\ tr$ is to define a message originated by an honest agent. Formally, regularOrig $m\ tr \equiv \forall A.$originates $A\ m\ tr \Longrightarrow A \notin$ bad. The predicate nonceDisj $m\ tr$ specifies that the nonces of message m are disjoint with any other messages occurring in the trace tr. Namely, if nonces of any message m' are not disjoint with those of m, then $m = m'$.

```
definition nonceDisj::"msg⇒ trace ⇒ bool"
      where nonceDisj m tr ≡∀ A M m'.
         Says A M m'∈(set tr)
         ∧ (noncesOf m' ∩ noncesOf m≠ ∅) ⟹ m'=m
```

We define single_valued r as $\forall\ x\ y.\ (x, y) \in r \longrightarrow (\forall\ z.\ (x, z) \in r \longrightarrow y = z)$. Obviously, if single_valued r, then a function f from the domain of r to range of r can be derived by $f\ x = y$ if $(x, y) \in r$; otherwise $f\ x = x$. If single_valued r^{-1} also holds, then such f is a bijection.

Next we define a set of special lists: distinctList. If $tr \in$ distinctList, $i, j <$ length tr, and $i \neq j$, then we have $tr_i \neq tr_j$. Here tr_i is the i-th element of the list tr. Namely, two elements of tr are different from each other.

```
inductive_set distinctList::('a list) set where
      nilDiff:  "[] ∈ distinctList"
      |consDiff:  "⟦tr ∈ distinctList;
         ∀ l.l∈(set tr) ⟶ l ≠ a⟧⟹ (a#tr) ∈ distinctList"
```

2.2. Intruder model

We discuss anonymity properties based on observations of the intruder. In this section, we explain our intruder model. Dolev-Yao intruder model [9] is considered standard in the field of formal symbolic analysis of security protocols. In this model the network is completely under the control of the intruder: all

messages sent on the network are read by the intruder; all received messages on the network are created or forwarded by the intruder; the intruder can also remove messages from the network. However, in the analysis of anonymity protocols, often a weaker attacker model is assumed – the intruder is *passive* in the sense that he observes all network traffic, but does not actively modify the messages or inject new messages. Therefore, we only need one kind of event Says A B x in our theory, which means that A sends a message x to B, and B receives the message. This semantics is subtly different from [24], where A intends to send a message x to B, but B does not necessarily receive the message. Besides, the intruder can analyze the messages that he has observed, which is modeled by the operator analz. In the later sections on case studies, we will point out that some anonymity properties cannot be kept if we have the Dolev-Yao intruder model instead.

Contrary to the intruder, the regular agents are not necessarily aware of all the events. We adopt the convention that they only see the events in which they are involved as either sender or receiver. According to the above arguments, we can formalize the notion of visible part of a trace.

```
view A [] =[]" |
view A ((Says A' B x)#evs) =
      if A = Spy then (Says A' B x)# evs
      else if (A'=A ∨ B=A) then ((Says A' B x) # (view A evs))
          else (view A evs)
```

3. Message Distinguishability

In this section, we focus on modeling the ability for an agent to distinguish two received messages based on his knowledge. In principle, an agent can uniquely identify any plain-text message he observes. Furthermore, an agent can distinguish any encrypted message for which he possesses the decryption key, or which he can construct himself. Formally, if m and m' are of different type of messages, for instance, if $m =$ Agent A and $m' =$ Nonce n, the agent can immediately tell the difference. If both m and m' are composed messages, namely, $m = \{\!|m_1, m_2|\!\}$ and $m' = \{\!|m'_1, m'_2|\!\}$, the agent can distinguish m and m' if he either distinguishes m_1 from m'_1 or m_2 from m'_2. If $m = \{\!|x|\!\}_{k_1}$ and $m' = \{\!|y|\!\}_{k_2}$, then the agent must use the knowledge Kn he possesses to decide whether the two messages are different. There are five cases as shown below:

1. Both k_1 and k_2 are in Kn, x and y are in Kn as well, and the agent can distinguish x and y, then he can tell the difference between m and m' as he knows that m and m' are different encrypted messages containing different plaintexts.
2. Both k_1 and k_2 are in Kn, x, y are in Kn as well, and the agent can distinguish k_1 and k_2 but not x and y, then he also can tell the difference between m and m' as he knows that m and m' are different messages encrypted by different keys.

3. Both x and k_1 are in Kn, and the agent knows that he can construct m from x and k_1. However, either y or k_2 is *not* in Kn. The agent can also tell the difference between m and m' as m can be constructed by himself, but m' cannot be constructed by himself.
4. If $k_1^{-1}, k_2^{-1} \in Kn$, and the agent can distinguish x and y, then he also can tell the difference between m and m' as he knows that m and m' can be decrypted into different messages by using k_1^{-1} and k_2^{-1}.
5. If k_1^{-1} is in Kn, and $k_1^{-1} \neq k_2^{-1}$, then there are two subcases, (1) either $k_2^{-1} \in Kn$, thus the agent can tell the difference between them as he knows that the two messages can be decrypted by using different keys; (2) or $k_2^{-1} \notin Kn$, thus the agent can also tell the difference between them as he knows that the m can be decrypted but m' cannot be decrypted.

We capture the above ideas by the following formalization in Isabelle/HOL.

```
definition basicDiff::  "msg⇒msg⇒bool"
where "basicDiff m m' ≡
      case m of (Agent a) ⇒ m ≠ m'
      | (Number n) ⇒ m ≠ m'
      | (Nonce n) ⇒ m ≠ m'
      | (Key k) ⇒ m ≠ m'
      | (MPair m1 m2) ⇒ ∀ m1' m2' . m' ≠ (MPair m1' m2')
      | (Crypt k n) ⇒ ∀ k' n' . m' ≠ (Crypt k' n')
inductive_set Diff::  "msg set ⇒ (msg×msg) set"
      for Kn::  "msg set" where
      basic:"⟦x∈Kn; y∈Kn; basicDiff x y⟧
      ⟹ (x,y)∈ Diff Kn"
      | MPLDiff:"⟦w∈Kn; z∈Kn; (x,y)∈Diff Kn⟧
      ⟹ (MPair x w, MPair y z)∈Diff Kn"
      | MPRDiff:"⟦w∈Kn; z∈Kn; (x,y)∈Diff Kn⟧
      ⟹ (MPair w x, MPair z y)∈Diff Kn"
      | CryptDiff1:"⟦(Key k1∈Kn); (Key k2∈Kn); (x,y)∈Diff Kn⟧
      ⟹ (Crypt k1 x, Crypt k2 y)∈Diff Kn"
      | CryptDiff2:"⟦x∈Kn; y∈Kn; (Key k1,Key k2)∈Diff Kn⟧
      ⟹ (Crypt k1 x, Crypt k2 y)∈Diff Kn"
      | CryptDiff3:"⟦y∉Kn∨Key k2 ∉ Kn; x∈Kn; Key k1 ∈ Kn; Crypt k2 y∈Kn⟧
      ⟹ (Crypt k1 x, Crypt k2 y)∈Diff Kn"
      | CryptDiff4:"⟦y∉Kn∨Key k2 ∉ Kn; x∈Kn; Key k1 ∈ Kn; Crypt k2 y∈Kn⟧
      ⟹ (Crypt k2 y, Crypt k1 x)∈Diff Kn"
      | DeCryptDiff1:"⟦(Crypt k1 x)∈Kn; (Crypt k2 y)∈Kn;
      (Key (invKey k1)∈Kn); (Key (invKey k2)∈Kn); (x,y)∈Diff Kn⟧
      ⟹ (Crypt k1 x, Crypt k2 y)∈Diff Kn"
      | DecryptDiff2:"⟦(Crypt k1 x)∈Kn; (Crypt k2 y)∈Kn;
      (Key (invKey k1))∈Kn; (Key (invKey k1))≠(Key (invKey k2))⟧
      ⟹ (Crypt k1 x, Crypt k2 y)∈Diff Kn"
      | DecryptDiff3:"⟦(Crypt k1 x)∈Kn; (Crypt k2 y)∈Kn;
      (Key (invKey k1))∈Kn; (Key (invKey k1))≠(Key (invKey k2))⟧
      ⟹ (Crypt k2 y, Crypt k1 x)∈Diff Kn"
```

Note that rules `CryptDiff3` and `CryptDiff4` are two symmetric subcases of case 3, and rules `DecryptDiff2` and `DecryptDiff3` are two subcases of case 5.

In this paper, when we discuss Diff Kn, we always assume that Kn is a closure set under the analz and then synth operators. Namely, $Kn = \mathsf{synth}\ (\mathsf{analz}\ Kn)$ for some message set Kn which is directly observed from network traffics.

Example 1. *Let* $m = \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ B}$, *and* $m' = \{\!|\mathsf{Nonce}\ n'|\!\}_{\mathsf{pubK}\ B}$, *and suppose* $Kn = \mathsf{synth}(\mathsf{analz}\{\mathsf{Key}\ (\mathsf{priK}\ B), m, m'\})$, *and* $n \neq n'$, *we have* $(m, m') \in \mathsf{Diff}\ Kn$ *by applying rule* `basic` *and rule* `CryptDiff`.

Example 2. *Let* $n'_0 \neq n$, $n'_0 \neq n'$, $n \neq n'$, $A \neq B$, $n_0 \neq n$, $n_0 \neq n'$, $m = \mathsf{Nonce}\ n$, $m' = \mathsf{Nonce}\ n'$, $m_0 = \{\!|m|\!\}_{\mathsf{pubK}\ B}$, $m'_0 = \{\!|m'|\!\}_{\mathsf{pubK}\ B}$, $m_1 = \{\!|\mathsf{Nonce}\ n_0, m_0|\!\}$, $m'_1 = \{\!|\mathsf{Nonce}\ n'_0, m'_0|\!\}$, $m_2 = \{\!|\mathsf{Agent}\ B, m_1|\!\}_{\mathsf{pubK}\ A}$, $m'_2 = \{\!|\mathsf{Agent}\ B, m'_1|\!\}_{\mathsf{pubK}\ A}$, *if* $Kn = \mathsf{synth}(\mathsf{analz}\ \{m_0, m'_0, m_2, m'_2, \mathsf{Key}\ (\mathsf{pubK}\ A), \mathsf{Key}\ (\mathsf{pubK}\ B), \mathsf{Key}\ (\mathsf{priK}\ B)\})$, *then we have* $(m_2, m'_2) \notin \mathsf{Diff}\ Kn$.

In Example 2, as priK B, m_0 and m'_0 are in Kn, therefore Nonce $n \in Kn$ and Nonce $n' \in Kn$. The conditions $n_0 \neq n$ and $n_0 \neq n'$ eliminate the possibility of the case when Nonce $n_0 \in Kn$. Similarly, we can derive that Nonce $n'_0 \notin Kn$.

We then introduce the notion of observational equivalence between messages which is naturally defined as the negation of message distinguishability. If an agent cannot distinguish two messages m and m', then the two messages are observationally equivalent to the agent.

```
msgEq::"msg set⇒msg⇒msg⇒bool"
         "msgEq Know m1 m2 ≡ (m1, m2)∉ Diff Know"
```

Obviously, observational equivalence between messages w.r.t. a knowledge set Kn is reflexive, symmetric and transitive.

Lemma 3. msgEq $Kn\ m\ m$

Lemma 4. msgEq $Kn\ m\ n \Longrightarrow$ msgEq $Kn\ n\ m$

Lemma 5. $[\![\mathsf{msgEq}\ Kn\ m_1\ m_2; \mathsf{msgEq}\ Kn\ m_2\ m_3]\!] \Longrightarrow \mathsf{msgEq}\ Kn\ m_1\ m_3$

4. Alignment between Two Message Sequences

In order to define observational equivalence between two traces (see Sect. 5), we first propose a requirement, called alignment, on two message sequences. The intuitive idea of our alignment requirement is that the relation, composed of corresponding message pairs in two message sequences, should be single-valued. For instance, there are two runs $msgSq$ and $msgSq'$ of a protocol, as shown in Tab. 1. Let $Kn = \mathsf{synth}(\mathsf{anlaz}\{m, m', m''\})$, $m \neq m'$. Even if we have msgEq $Kn\ m\ m'$ and msgEq $Kn\ m\ m''$, $msgSq$ and $msgSq'$ should still be different from an observer's view, because the same message m occurs twice in $msgSq$ while two different messages m' and m'' occur in the corresponding positions of $msgSq'$. Alignment requires that a message should have only one

Table 1: Two non-alignment message sequences

$msgSq$	$msgSq'$
m	m'
m	m''

Table 2: Two non-alignment message sequences

$msgSq$	$msgSq'$
$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}\}_{\mathsf{priK}\ M}$	$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}\}_{\mathsf{priK}\ M}$
$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}\}_{\mathsf{priK}\ M}$	$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}\}_{\mathsf{priK}\ M}$
$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}$	$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}$
$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}$	$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}$

interpretation when we map messages from a message sequence to the other message sequence.

Furthermore, single-valued requirement should remain valid after applying the analyzing operation (e.g., decryption and seperation) and synthesizing operation (e.g., encryption and concatenation) pairwisely on the message pairs in the two message lists of the two message sequences. From Examples 6 to 8, we use two message sequences $msgSq$ and $msgSq'$ to explain the above two requirements. Below n_0, n_1, n_0', n_1' are pairwise different nonces.

Example 6. *If* priK B *and* priK B' *are not compromised, then* $msgSq$ *and* $msgSq'$ *as shown in Tab. 2 are different w.r.t. an intruder as the intruder can decrypt the first and second messages and compare them with the third and fourth messages in the above message sequences. (After applying the decryption operation to the first messages pairwise in the two message sequences, the intruder obtains a new pair* $(\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}, \{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'})$*. But this pair and* $(\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}, \{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B})$ *contradicts with the single-valued requirement.)*

Example 7. *If* priK B *and* priK B' *and* priK M *are not compromised, then* $msgSq$ *and* $msgSq'$ *as shown in Table 3 are different w.r.t. an intruder as the intruder can encrypt the third and fourth messages and compare them with the first and second messages in the above two sequences. (After applying the encryption operation to third messages pairwise in the two message sequences, the intruder obtains a new pair* $(\{\!\{\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}\}\!\}_{\mathsf{pubK}\ M}, \{\!\{\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}\}\!\}_{\mathsf{pubK}\ M})$*. But this pair and* $(\{\!\{\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}\}\!\}_{\mathsf{pubK}\ M}, \{\!\{\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}\}\!\}_{\mathsf{pubK}\ M})$ *contradicts with the single-valued requirement.)*

Example 8. *If* priK B *and* priK B' *and* priK M *are not compromised,* $msgSq$ *and* $msgSq'$ *as shown in Table 4 should be equivalent w.r.t. an intruder as all the messages cannot be analyzed and the linkage of messages in a trace cannot be established.*

Table 3: Two non-alignment message sequences

$msgSq$	$msgSq'$
$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}\}_{\mathsf{pubK}\ M}$	$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}\}_{\mathsf{pubK}\ M}$
$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}\}_{\mathsf{pubK}\ M}$	$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}\}_{\mathsf{pubK}\ M}$
$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}$	$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}$
$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}$	$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}$

Table 4: Two alignment message sequences

$msgSq$	$msgSq'$
$\{\!\{\mathsf{Nonce}\ n_0, \mathsf{Agent}\ B, \{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}\}_{\mathsf{pubK}\ M}$	$\{\!\{\mathsf{Nonce}\ n_1, \mathsf{Agent}\ B', \{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}\}_{\mathsf{pubK}\ M}$
$\{\!\{\mathsf{Nonce}\ n_1, \mathsf{Agent}\ B', \{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}\}_{\mathsf{pubK}\ M}$	$\{\!\{\mathsf{Nonce}\ n_0, \mathsf{Agent}\ B, \{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}\}_{\mathsf{pubK}\ M}$
$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}$	$\{\!\{\mathsf{Nonce}\ n_0'\}\!\}_{\mathsf{pubK}\ B}$
$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}$	$\{\!\{\mathsf{Nonce}\ n_1'\}\!\}_{\mathsf{pubK}\ B'}$

More formally, we first inductively define two more operators analz_pairs and synth_pairs to formalize the above pairwise analyzing and synthesizing operations on the message pairs between two sets of message pairs.

```
inductive_set analz_pairs::"(msg×msg) set⇒msg set⇒(msg×msg) set"
for r ::"(msg×msg) set" and Kn::"msg set"
where rAtom [intro]:  "⟦(x,y):r⟧⟹ (x, y)∈ analz_pairs r Kn"
| MPairL_closure [intro]:  "⟦({|x,y|},{|x',y'|}) ∈analz_pairs r Kn⟧
      ⟹(x,x')∈analz_pairs r Kn"
| MPairR_closure [intro]:  "⟦({|x,y|},{|x',y'|}) ∈analz_pairs r Kn⟧
      ⟹(y,y')∈analz_pairs r Kn"
| deCrypt_closure [intro]:  "⟦(Crypt k x,Crypt k x')∈analz_pairs r Kn;
      Key (invKey k)∈Kn ⟧ ⟹ (x,x')∈analz_pairs r Kn"
```

```
inductive_set synth_pairs::"(msg ×msg) set⇒msg set⇒(msg×msg) set"
for r ::"(msg× msg) set" and Kn::"msg set"
where basicAtom [intro]:  "⟦x∈Kn; isAtom x⟧⟹ (x, x)∈ synth_pairs r Kn"
| rAtom [intro]:  "⟦(x,y)∈r ⟧⟹ (x, y)∈ synth_pairs r Kn"
| MPair_closure [intro]:  "⟦(x,x')∈synth_pairs r Kn;
      (y,y')∈ synth_pairs r Kn⟧⟹ (⟦x,y⟧,⟦x',y'⟧)∈ synth_pairs r Kn"
| Crypt_closure [intro]:  "⟦(x,x')∈synth_pairs r Kn; Key k∈Kn⟧
      ⟹ (Crypt k x, Crypt k x')∈synth_pairs r Kn"
```

The following lemma gives a sufficient condition for the existence of a function mapping which is naturally derived from synth_pairs $r\ Kn$ provided that r is single-valued.

Lemma 9. ⟦single_valued r; $(x, y) \in$ (synth_pairs $r\ Kn$);
$(x', y') \in$ synth_pairs $r\ Kn$; $x = x'$;
$\forall\ m\ m'\ m''.(m, m'') \in$ synth_pairs $r\ Kn \longrightarrow ((m, m') \in r \longrightarrow m' = m'')$ ⟧
$\Longrightarrow y = y'$

Note that we cannot establish a similar result for the analz_pairs operator. For instance, let $r = \{(\{\!|\mathsf{Nonce}\ n, \mathsf{Nonce}\ n|\!\}, \{\!|\mathsf{Nonce}\ n, \mathsf{Nonce}\ n'|\!\})\}$. It is easy to verify that analz_pairs $r\ Kn$=$\{(\mathsf{Nonce}\ n, \mathsf{Nonce}\ n), (\mathsf{Nonce}\ n, \mathsf{Nonce}\ n')\}$. We have single_valued r and $\forall\ m\ m'\ m''.(m, m'') \in$ analz_pairs $r\ Kn \longrightarrow (m, m') \in r \longrightarrow m' = m''$. But analz_pairs $r\ Kn$ is not single_valued.

5. Observational Equivalence between Traces

Now we can lift observational equivalence to traces with the concepts of observational equivalence between messages and alignment between two message sequences: two sequences of messages in two traces look the same to an observer if a message in one sequence is observationally equivalent to the corresponding message in the other sequence w.r.t. the knowledge which the observer has obtained from the two traces. Besides the requirement on the message parts of the two traces, we require that the sender and receiver of an event in one trace is the same as those of the corresponding event in the other trace. For events ev_1 and ev_2, we define SRMatch $ev_1\ ev_2 \equiv$ (sender ev_1 = sender ev_2) $\wedge$ (receiver ev_1 = receiver ev_2). For two traces tr and tr', SRMatchL $tr\ tr' \equiv$ length tr = length $tr' \wedge \forall\ i.i <$ length $tr \longrightarrow$ SRMatch $tr_i\ tr'_i$. The predicate SRMatchL $tr\ tr'$ means that each event tr_i has the same sender and receiver as its corresponding event tr'_i and the two traces have the same length.

Two traces tr and tr' are observationally equivalent, written as $tr \approx_A tr'$, if the following conditions are satsified:

- tr and tr' have the same length; and for all events in tr_i, the senders and receivers of tr_i are the same as those of tr'_i.
- msgPart tr_i and msgPart tr'_i are observationally equivalent w.r.t. the knowledge obtained after observing the two traces.
- single_valued r and single_valued r^{-1} guarantee that an agent cannot reinterpret any event differently, where r (r^{-1}) is the sequence of message paris obtained from tr and tr' (tr' and tr) after applying the operations analz_pairs and synth_pairs.

The corresponding formalization in Isabelle/HOL is given below.

```
definition obsEquiv::"agent⇒trace⇒trace⇒bool"
      where "obsEquiv A tr tr'≡
         let vtr=view A tr in
         let vtr'=view A tr' in
         let msgSq=map msgPart vtr in
         let msgSq'=map msgPart vtr' in
         (set msgSq)=(set msgSq') ∧ length vtr=length vtr'∧
         SRMatchL vtr vtr'∧
         (let H=set (zip msgSq msgSq') in
          let Kn=synth (analz (knows A vtr)) in
         (∀x y.  (x,y)∈ H ⟶ msgEq Kn x y)∧
          (let r=synth_pairs (analz_pairs H Kn) Kn in
          (single_valued r ∧ single_valued (r⁻¹)))"
```

Remark 10. *In the work of Garcia et al. [11], a reinterpretation function between two message sequences is used as a underlining concept. However, no one has formally argued when such a function exists and how it can be derived. In our work, the alignment requirement between the two message sequences gives a sufficient condition for the existence of a reinterpretation function. Moreover, the two operators* analz_pairs *and* synth_pairs *give a mechanical way to derive the reinterpretation function. Note that if both single_valued r and single_valued (r^{-1}), we can naturally construct a bijection function between the domain of r to its range.*

6. Epistemic Operators and Anonymity Properties

Using the observational equivalence relations over a trace set of possible worlds, we can formally introduce epistemic operators [11] as follows:

```
constdefs box::"agent⇒trace⇒trace set⇒
         assertOfTrace⇒bool"
      "box A tr trs Assert≡
         ∀tr'.tr'∈trs⟶obsEquiv A tr tr' ⟶(Assert tr')"

constdefs diamond::"agent⇒trace⇒trace set⇒
         assertOfTrace⇒bool"
      "diamond A tr trs Assert≡
         ∃tr'.tr'∈trs ∧obsEquiv A tr tr'
         ∧(Assert tr')"
```

For notation convenience, we write $tr \models \Box\ A\ trs\ \varphi$ for box $A\ tr\ trs\ \varphi$, and $tr \models \Diamond\ A\ trs\ \varphi$ for diamond $A\ tr\ trs\ \varphi$. Note that φ is a predicate on a trace. Intuitively, $tr \models \Box\ A\ trs\ \varphi$ means that for any trace tr' in trs, if tr' is observationally equivalent to tr for agent A, then tr' satisfies the assertion φ. On the other hand, $tr \models \Diamond\ A\ trs\ \varphi$ means that there is a trace tr' in trs, tr' is observationally equivalent to tr for agent A and tr' satisfies the assertion φ. Now we can formulate some information hiding properties in our epistemic language. We use the standard notion of an anonymity set: it is a collection of agents among which a given agent is not identifiable. The larger this set is, the more anonymous an agent is.

6.1. Sender anonymity

Suppose that tr is a trace of a protocol in which a message m is originated by some agent. We say that tr provides sender anonymity w.r.t. the anonymity set AS and a set of possible runs in the view of B if it satisfies:

```
constdefs senderAnomity::"agent set⇒agent⇒msg⇒
         trace⇒trace set⇒bool"
      "senderAnomity AS B m tr trs≡ (∀X.X:AS⟶
         tr ⊨◇B trs (originates X m))"
```

Here, AS is the set of agents who are under consideration, and trs is the set of all the traces which B can observe. Intuitively, this definition means that each agent in AS can originate m in a trace of trs. Therefore, this means that B cannot be sure of anyone who originates this message.

6.2. Unlinkability

We say that a trace tr provides unlinkability for user A and a message m w.r.t. the anonymity set AS if

```
constdefs unlinkability::"agent set⇒agent⇒msg⇒
         trace⇒trace set⇒bool"
      "unlinkability AS A m tr trs≡
         (let P= λX m' tr. originates X m' tr in
            (¬(tr ⊨□ Spy trs (P A m)))
         ∧ senderAnomity AS A m tr trs
```

where the left side of the conjunction means that the intruder is not certain whether A has sent the message m, while the right side means that every other user could have sent m.

7. Case Study I: Crowds

The Crowds system [26] is a system for performing anonymous web transactions based on the idea that anonymity can be provided by hiding in a crowd. For simplicity reasons, we only model the request part as specified in [11]: when an agent wants to send a request to a server, he randomly selects a user from a crowd of users and asks this user to forward the request for him to the server; and this user then either forwards the request to the server, or selects another random user from the crowd to do the forwarding. The specification of Crowds is shown as below:

```
inductive_set Crowds::  trace set where
CrowdsNil:  [] ∈ (Crowds)
| CrowdsInit:  ⟦tr∈Crowds; Nonce n∉(used tr);R≠Server; A≠Server⟧
      ⟹Says A R {|Agent Server, Nonce n|}#tr∈Crowds
| CrowdsRelay:  ⟦tr∈Crowds; Says R R' {|Agent Server, Nonce n|}∈set tr;
         R'≠Server; R''≠Server⟧
      ⟹Says R' R'' {|Agent Server, Nonce n|}#tr∈Crowds
| CrowdsSend:  ⟦tr∈Crowds; Says R R' {|Agent Server, Nonce n|}∈set tr;
         R'≠Server; ∀R'. (Says R' Server (Nonce n))∉set tr ⟧
      ⟹Says R' Server (Nonce n)#tr∈Crowds
```

In the above formalization, rule `crowdNil` specifies an empty trace. The other rules specify trace's extension with protocol steps. More precisely,

- rule `CrowdsInit` models that an agent A, who is not the Server, originates a requests. Here, we model new requests as fresh nonces. The agent

randomly selects a user R from a crowd of users and asks this user to forward the request for him to the Server;

- rule `CrowdsRelay` specifies that a relay R' selects another random user R'' again from the crowd to do the forwarding. Here, we simply require that R'' is not the Server;
- rule `CrowdsSend` models that a relay R' forwards the request to the Server. Here, the requirement $\forall R'.(\mathsf{Says}\ R'\ \mathsf{Server}\ (\mathsf{Nonce}\ n) \notin \mathsf{set}\ tr$ specifies that no other user has sent the request to the Server before.

The following lemma simply states the fact that a request forwarded to the server must be initiated by an agent before.

Lemma 11. $[\![tr \in \mathsf{Crowds}; \mathsf{Says}\ R\ \mathsf{Server}(\mathsf{Nonce}\ n) \in \mathsf{set}\ tr]\!] \Longrightarrow$
$\exists A\ B.\mathsf{Says}\ A\ B\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\} \in \mathsf{set}\ tr$

Suppose that there exists an event $\mathsf{Says}\ A\ B\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}$ occurring in a trace tr, then there exist two subtrace tr_1 and tr_2, two agents A' and B' such that $tr = tr_1@(\mathsf{Says}\ A'\ B'\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\#tr_2)$ and the subtrace tr_2 does not contain any event whose message is of the form $\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}$. We can prove it simply by induction on tr.

Lemma 12. $[\![\mathsf{Says}\ A\ B\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\} \in \mathsf{set}\ tr]\!] \Longrightarrow$
$\exists tr_1\ tr_2\ A'\ B'.tr = tr_1@(\mathsf{Says}\ A'\ B'\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\#tr_2)$
$\wedge(\forall\ A\ B.\mathsf{Says}\ A\ B\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\} \notin \mathsf{set}\ tr_2)$

By the above two lemmas, and since $\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}$ does not occur in tr_2, therefore we can know that the agent A' originates the nonce n.

Lemma 13. $[\![\ tr \in \mathsf{Crowds}; \mathsf{Says}\ R\ \mathsf{Server}\ (\mathsf{Nonce}\ n) \in \mathsf{set}\ tr]\!] \Longrightarrow$
$\exists A.\mathsf{originates}\ A\ (\mathsf{Nonce}\ n)\ tr$

Assume that $tr = tr_1@\mathsf{Says}\ A'\ B'\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\#tr_2$ is a trace in Crowds, and the message $\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}$ does not occur in tr_2. Namely, A' is the agent who originates the request $\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}$. We can add a new event $\mathsf{Says}\ A\ A'\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}$ before tr_2. Then the new trace $tr_1@(\mathsf{Says}\ A'\ B'\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\#\mathsf{Says}\ A\ A'\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\#tr_2$ is still a valid trace in Crowds. This is formulated in the next lemma, which is crucial to prove sender anonymity for agent A' as another agent A seems possible for the observer to initiate the request as well. This is due to the fact that the newly constructed trace is valid in the Crowds system.

Lemma 14. $[\![\ tr \in \mathsf{crowd};\ tr = tr_1@(Says\ A'\ B'\ \{\!|\mathsf{Agent}\ Server, \mathsf{Nonce}\ n|\!\}\#tr_2);$
$(\forall A\ B.\mathsf{Says}\ A\ B\{\!|\mathsf{Agent\ Server}, Nonce\ n|\!\} \notin \mathsf{set}\ tr_2)\ ]\!] \Longrightarrow$
$tr_1@(\mathsf{Says}\ A'\ B'\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\#\mathsf{Says}\ A\ A'\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\#tr_2)$
$\in \mathsf{Crowds}$

Suppose that the Server receives a request (identified by a nonce Nonce n), then the Server cannot be sure of which agent originates the request. That is to say, the sender anonymity holds for the Server w.r.t any anonymity agent set not containing the Server.

Lemma 15. $[\![tr \in \mathsf{Crowds}; \mathsf{Says}\ R\ \mathsf{Server}\ (\mathsf{Nonce}\ n) \in \mathsf{set}\ tr]\!] \Longrightarrow$
$\mathsf{senderAnonymity}\ \{A.A \neq \mathsf{Server}\}\ \mathsf{Server}\ (\mathsf{Nonce}\ n)\ tr\ \mathsf{Crowds}$

Proof. By unfolding the definition of senderAnonymity, for any agent X such that $X = \mathsf{Server}$, we need to find a trace tr' such that $tr' \in \mathsf{Crowds}$, $\mathsf{obsEquiv}\ \mathsf{Server}\ tr\ tr'$ and $\mathsf{originates}\ X\ (\mathsf{Nonce}\ n)\ tr'$. By $\mathsf{Says}\ R\ \mathsf{Server}\ (\mathsf{Nonce}\ n) \in \mathsf{set}\ tr$ and Lemma 13, there exists an agent A such that $\mathsf{originates}\ A\ (\mathsf{Nonce}\ n)\ tr$. There are two cases:
(1)$A = X$. Then we simply let $tr' = tr$.
(2)$A \neq X$. By $\mathsf{Says}\ R\ \mathsf{Server}\ (\mathsf{Nonce}\ n) \in \mathsf{set}\ tr$, and Lemma 11, there exist agents A and B such that $\mathsf{Says}\ A\ B\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\} \in \mathsf{set}\ tr$. Then by Lemma 12, there exist tr_1, tr_2, A', and B' such that tr can be transformed into $tr_1@\mathsf{Says}\ A'\ B'\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\#tr2$, and we have the fact that $\forall C\ D.\mathsf{Says}\ C\ D\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\} \notin \mathsf{set}\ tr_2$. Then we can construct tr' as $tr_1@\mathsf{Says}\ A'\ B'\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\ (\mathsf{Says}\ X\ A'\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\})\#tr2$. By Lemma 14, we have $tr' \in \mathsf{Crowds}$. By Lemma 13, $\mathsf{originates}\ X\ (\mathsf{Nonce}\ n)\ tr'$. Obviously, from the inductive definition of Crowds, we have $A' \neq \mathsf{Server}$. It is easy to verify that $\mathsf{view}\ \mathsf{Server}\ tr' = \mathsf{view}\ \mathsf{Server}\ tr$. Then we can derive $\mathsf{obsEquiv}\ \mathsf{Server}\ tr\ tr'$. ■

The sender anonymity comes from the local view of the agent Server, and the nondeterministic choice of a relay who either forwards a request again or directly sends the request to the Server. However, for the Spy, who observes the global network traffic, the sender anonymity does not hold. Namely, the Spy can be sure of the agent who originates a request. This can be formalized and proved as Lemma 16.

Lemma 16. $[\![\ tr \in \mathsf{Crowds}; tr = tr_1@(\mathsf{Says}\ A'\ B'\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\}\#tr_2);$
$(\forall A\ B.\mathsf{Says}\ A\ B\{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\} \notin \mathsf{set}\ tr_2)]\!] \Longrightarrow$
$\square\ \mathsf{Spy}\ tr\ \mathsf{Crowds}\ (\mathsf{originates}\ A'\ (\mathsf{Nonce}\ n))$

8. Case Study II: Onion Routing

Onion Routing [12, 29] provides both sender and receiver anonymity for communication over the Internet and servers as the basis of the Tor network [8]. Its main idea is based on Chaum's mix cascades [3] that messages in Onion Routing have a layered encryption (thus called *onions*) and travel from source to destination via a sequence of proxies (called *onion routers*). Each onion router can decrypt (or peel) one layer of a received message and forward the remainder to the next onion router. In order to disguise the relations between incoming and outgoing messages, an onion routers collect incoming messages until it has received k messages, and then permutes the messages and sends in batch to their intended receivers.

8.1. Modeling Onion Routing

In this paper, we model a simplified ed version of Onion Routing with only one onion router as done in [11]. We assume a set of users AS and one router M, with $M \notin AS$. We also assume that each agent can send a message before the router M launches a batch of forwarding process, and the router does not accept any message when it is forwarding.

```
inductive_set oneOnionSession::  "nat⇒agent⇒trace set"
for i::"nat" and M::"agent" where
      onionNil:  "[]∈ (oneOnionSession i M)"
      | onionCons1:  "⟦tr∈(oneOnionSession i M);X≠M;Y≠M;
         Nonce n0∉(used tr);Nonce n∉(used tr); length tr<i⟧⟹
         Says X M (Crypt (pubK M)
            {|Nonce n0,Agent Y,Crypt (pubK Y) (Nonce n)|})
         #tr ∈oneOnionSession i M"
      | onionCons2:  "⟦tr∈(oneOnionSession i M);X≠M;
         Nonce n∉(used tr);length tr<i⟧⟹
         Says X M (Crypt (pubK M) (Nonce n))
         #tr ∈oneOnionSession i M"
      | onionCons3:  "⟦tr∈(oneOnionSession i M);length tr≥i;
         Says M Y (Crypt (pubK Y) (Nonce n))∉(set tr)⟧⟹
         Says M Y (Crypt (pubK Y) (Nonce n))
         #tr ∈oneOnionSession i M"
```

In the above specification of Onion Routing, there are four induction rules. Rule `onionNil` specifies an empty trace. The other rules specify trace's extension with protocol steps. The ideas behind these induction rules (`onionCons1`, `onionCons2`, `onionCons3`) are explained as follows.

- If the length of the current trace is less than i, namely, M is still in a receiving status, X (or Y) and M are distinct, and both n_0 and n are fresh, we can add an event $\mathsf{Says}\ X\ M\ \{|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{|\mathsf{Nonce}\ n|\}_{\mathsf{pubK}\ Y}|\}_{\mathsf{pubK}\ M}$. This step means that X sends a message to M which will be peeled and forwarded to Y by M.

- If the length of the current trace is less than i, X and M are distinct, and n is fresh, then we can add an event $\mathsf{Says}\ X\ M\{|\mathsf{Nonce} N|\}_{\mathsf{pubK}\ M}$. This means that X sends a dummy message to M which will be simply discarded later.

- If the length of the current trace is greater than or equal to i meaning that M is in a forwarding status, and if a received message of the form $\{|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{|\mathsf{Nonce}\ n|\}_{\mathsf{pubK}\ Y}|\}_{\mathsf{pubK}\ M}$ has not been forwarded by the router yet, then we can add an event $\mathsf{Says}\ M\ Y\ \{|\mathsf{Nonce}\ n|\}_{\mathsf{pubK}\ Y}$. This step means that the router M forwards the peeled message to Y.

In the following analysis, our intruder model is passive in the sense that the spy will not modify the network traffic. An active intruder can easily infer the receiver of a message m forwarded to some agent. He only needs to intercept

any other message except the message m, and replace them by faked dummy messages. Because all dummy messages will be discarded by the router, and only m will be peeled and forwarded to the intended receiver.

8.2. An overview our proof strategy

In the following sections, we will formalize and prove the anonymity properties of Onion Routing. Due to the complexity of the epistemic operators in anonymity definitions, the proof is rather complicated. We illustrate the overview of our formalization and the corresponding proof steps.

We will formalize the sender anonymity and unlinkabilty of Onion Routing in the view of a Spy for a trace tr w.r.t. a set of honest agents and all possible traces. According to the definition of epistemic operators in the definition of sender anonymity and unlinkability, we need to construct another trace tr' which satisfies two conditions:

(1) tr' is still an onion routing protocol trace, namely $tr' \in \mathsf{oneOnionSession}\ i\ M$.
(2) tr' is observation equivalent to tr. That is to say, $\mathsf{obsEquiv}\ \mathsf{Spy}\ tr\ tr'$. In order to show this, by the definition of obsEquiv, we need to prove four subcases. The first two subcases are straightforward, but the latter two are rather difficult: (i) $\mathsf{msgPart}\ tr_i$ and $\mathsf{msgPart}\ tr'_i$ for any $i < \mathsf{length}\ tr$ are observationally equivalent w.r.t. the knowledge obtained after observing the two traces; (ii) the alignment requirements $\mathsf{single\_valued}\ r$ and $\mathsf{single\_valued}\ r^{-1}$ where r is the sequence of message paris obtained from tr and tr' after applying the operations analz_pairs and synth_pairs.

Sect. 8.4 formally introduces a function *swap ma mb tr*, which servers the aim of constructing such an trace tr'. Here *ma*, *mb* are the messages sent to the router in the trace tr. Sect. 8.4.1 gives its formal definition and proves simple correspondence properties of the swap operator. Sect. 8.4.2 proves the first condition (1). Sect. 8.4.3 devotes to the proof of (2-ii), and Sect. 8.4.4 proves (2-i), then finishes the proof of (2). In order to prove (2-i), we need to prove properties such as secrecy and correspondence properties of Onion Routing, which are discussed in Sect. 8.3. After these, we finish the proofs of the two anonymity properties in Sect. 8.5.

8.3. Properties on protocol sessions

As mentioned before, whether two traces are observationally equivalent for an agent depends on the knowledge of the agent after his observation of the two traces. Therefore, we need to discuss some properties on the knowledge of the intruder. They are secrecy properties,and some regularity on the correspondence of the events in one protocol session of Onion Routing.

8.3.1. Correspondence properties

The following lemma is about the correspondence of two events in a trace tr. If the router M forwards a message $\{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}$, then there must exist an agent A who has sent a message $\{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$ using some nonce n_0.

Lemma 17. $[\![tr \in \mathsf{oneOnionSession}\ i\ M;\ \mathsf{Says}\ M\ B\ \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y} \in \mathsf{set}\ tr]\!]$
$\Longrightarrow \exists n_0\ A.\mathsf{Says}\ A\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M} \in \mathsf{set}\ tr$

If $\{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}$ is a submessage of a message which A sends to the router M, then $\{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}$ is originated by A.

Lemma 18. $[\![tr \in \mathsf{oneOnionSession}\ i\ M;\ ma' = \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y};$
$\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; ma' \sqsubset ma]\!] \Longrightarrow \mathsf{originates}\ A\ ma'\ tr$

8.3.2. Uniqueness properties

Since an agent is required to originate fresh nonces when he sends a message to the router, therefore if two events where agents send a message to the router M, either two events are exactly the same, or nonces used in the two events are disjoint.

Lemma 19. $[\![tr \in \mathsf{oneOnionSession}\ i\ M;\ \mathsf{Says}\ X\ M\ ma; \mathsf{Says}\ Y\ M\ mb]\!]$
$\Longrightarrow (X = Y \wedge ma = mb) \vee (\mathsf{noncesOf}\ ma \cap \mathsf{noncesOf}\ mb) = \emptyset$

From Lemma 19, we can easily derive that once a nonce n occurs in a message sent by an agent X, then another agent Y cannot originate a message containing the same nonce n.

Lemma 20. $[\![tr \in \mathsf{oneOnionSession}\ i\ M; \mathsf{Says}\ X\ M\ ma;\ X \neq Y;$
$\{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y} \sqsubset ma]\!] \Longrightarrow \neg\mathsf{originates}\ Y\ (\{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y})\ tr$

The message of each event in a trace of the protocol is unique, namely two messages in two events in this trace are different.

Lemma 21. $[\![tr \in \mathsf{oneOnionSession}\ i\ M]\!] \Longrightarrow \mathsf{map}\ \mathsf{msgPart}\ tr \in \mathsf{distinctList}$

With the above lemma, we can derive that the relation $(\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)\ sq')$ must be $\mathsf{single\_valued}$ if tr is in a trace of Onion Routing.

Lemma 22. $[\![tr \in \mathsf{oneOnionSession}\ i\ M]\!] \Longrightarrow \mathsf{single\_valued}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)\ sq')$

8.3.3. Secrecy properties

First we need to introduce a new predicate:

$\mathsf{nonLeakMsg}\ m\ M \equiv$
$\forall\ B\ n_0\ n.(m = (\mathsf{Crypt}\ (\mathsf{pubK}\ M)\{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ B, \mathsf{Crypt}\ (\mathsf{pubK}\ B)(\mathsf{Nonce}\ n)|\!\}))$
$\longrightarrow (B \notin \mathsf{bad}\ \vee\ n_0\ \neq\ n)$

Formally, $\mathsf{nonLeakMsg}\ m\ M$ specifies that if message m is of the form $\mathsf{Crypt}\ (\mathsf{pubK}\ M)\{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ B, \mathsf{Cry}$ then either $B \notin \mathsf{bad}$ or $n_0 \neq n$. This definition specifies a non-leakage condition of nonce part n_0 in a message of the form $\mathsf{Crypt}\ (\mathsf{pubK}\ M)\{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ B, \mathsf{Crypt}\ (\mathsf{pubK}\ B)(\mathsf{Nonce}\ n)|\!\}$ which is sent to the router even if whose nonce part n will be forwarded to a spy. The following lemma will explain the intuition behind this definition.

If both the router M and an agent B are honest, and B sends a message $ma = \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$ to M, and $\mathsf{nonLeakMsg}\ ma\ M$ also holds, then $\mathsf{Nonce}\ n_0$ cannot be analyzed by the intruder.

Lemma 23. $[\![tr \in \mathsf{oneOnionSession}\ i\ M; M \notin \mathsf{bad}; B \notin \mathsf{bad};$
$\mathsf{Says}\ B\ M\ ma \in tr;$
$ma = \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\}_{\mathsf{pubK}\ Y}|\}_{\mathsf{pubK}\ M}; \mathsf{nonLeakMsg}\ ma\ M]\!]$
$\Longrightarrow \mathsf{Nonce}\ n_0 \notin \mathsf{analz}\ (\mathsf{knows}\ \mathsf{Spy}\ tr)$

Similarly, provided that both M and B are honest, and B sends a dummy message $\{\!|\mathsf{Nonce}\ n_0|\}_{\mathsf{pubK}\ M}$ to M, then the intruder cannot know $\mathsf{Nonce}\ n_0$.

Lemma 24. $[\![tr \in \mathsf{oneOnionSession}\ i\ M; \mathsf{Says}\ B\ M\{\!|\mathsf{Nonce}\ n_0|\}_{\mathsf{pubK}\ M} \in tr; M \notin \mathsf{bad}; B \notin \mathsf{bad}]\!] \Longrightarrow \mathsf{Nonce}\ n_0 \notin \mathsf{analz}\ (\mathsf{knows}\ \mathsf{Spy}\ tr)$

8.4. Message swapping

By its definition, to prove sender anonymity of an agent X in a trace tr, we need to show the existence of an observationally equivalent trace tr'. In this section, we present a method for the construction of an observationally equivalent trace.

8.4.1. Formal definition of swap *function*

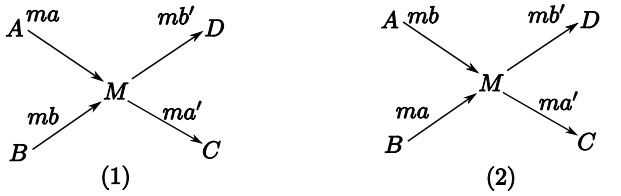
We define a function $\mathsf{swap}\ ma\ mb\ tr$, which returns another trace tr' satisfying the following conditions: (1) the sender and receiver of any event in trace tr' are the same as in the corresponding event in tr; (2) the message of any event in tr' is swapped as mb if the message of the corresponding event in tr is ma; (3) the message of any event in tr' is swapped as ma if the message of the corresponding event in tr is mb; (4) otherwise the message is kept unchanged.

```
consts swap::"msg⇒msg⇒trace⇒trace"
primrec "swap ma mb [] =[]"
swap ma mb (ev#tr)=
      case ev of Says A0 M0 ma0) ⇒
         (if (ma0=ma)
         then Says A0 M0 mb# swap ma mb tr)
         else if (ma0=mb)
         then Says A0 M0 ma# swap ma mb tr
         else ev# (swap ma mb tr))
```

For a trace tr of Onion Routing, Fig. 1 illustrates the correspondence between tr and the function $\mathsf{swap}\ ma\ mb\ tr$. In session 1, agent A (B) communicates with C (D), while agent A (B) communicates with D (C) in session 2. The correspondence between tr and $\mathsf{swap}\ ma\ mb\ tr$ is formalized as the lemma below.

Lemma 25. *Let tr be a trace.*

1. $[\![(m_1, m_2) \in \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)\ (\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)))]\!]$
   $\Longrightarrow m_1 = m_2 \vee (m_1, m_2) = (ma, mb) \vee (m_1, m_2) = (mb, ma)$
2. $\mathsf{sendRecvMatchL}\ tr\ (\mathsf{swap}\ ma\ mb\ tr)$
3. $\mathsf{length}\ (\mathsf{swap}\ ma\ mb\ tr) = \mathsf{length}\ tr$
4. $\mathsf{set}\ (\mathsf{map}\ \mathsf{msgpart}\ tr) = \mathsf{set}\ (\mathsf{map}\ \mathsf{msgpart}\ (\mathsf{swap}\ mb\ ma\ tr))$
5. $\mathsf{swap}\ ma\ mb\ tr = \mathsf{swap}\ mb\ ma\ tr$



$ma = \{\!|\mathsf{Nonce}\ na, \mathsf{Agent}\ C, \{\!|\mathsf{Nonce}\ na'|\!\}_{\mathsf{pubK}\ C}|\!\}_{\mathsf{pubK}\ M}, ma' = \{\!|\mathsf{Nonce}\ na'|\!\}_{\mathsf{pubK}\ C}$

$mb = \{\!|\mathsf{Nonce}\ nb, \mathsf{Agent}\ D, \{\!|\mathsf{Nonce}\ nb'|\!\}_{\mathsf{pubK}\ D}|\!\}_{\mathsf{pubK}\ M}, mb' = \{\!|\mathsf{Nonce}\ nb'|\!\}_{\mathsf{pubK}\ D}$

Figure 1: An illustration of function swap.

6. $[\![(\mathsf{Says}\ X\ M\ ma \in \mathsf{set}\ tr)]\!] \Longrightarrow \mathsf{Says}\ X\ M\ mb \in \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr))$
7. $[\![(\mathsf{Says}\ X\ M\ mb \in \mathsf{set}\ tr)]\!] \Longrightarrow \mathsf{Says}\ X\ M\ ma \in \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr))$
8. $[\![m \neq ma; m \neq mb; (\mathsf{Says}\ X\ M\ m) \in \mathsf{set}\ tr]\!]$
   $\Longrightarrow (\mathsf{Says}\ X\ M\ m \in \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr))$
9. $[\![m \neq ma; m \neq mb; (\mathsf{Says}\ X\ M\ m) \notin \mathsf{set}\ tr]\!]$
   $\Longrightarrow (\mathsf{Says}\ X\ M\ m \notin \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr))$
10. $[\![\mathsf{Says}\ A\ M\ ma \in tr; \mathsf{Says}\ B\ M\ mb \in tr;$
    $\forall ev.ev \in tr \longrightarrow (\exists\ A'\ B'\ m.\ ev = \mathsf{Says}\ A'\ B'\ m)]\!]$
    $\Longrightarrow \mathsf{map}\ msgPart\ tr = \mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)$
11. $[\![\mathsf{Says}\ A\ M\ ma \in tr; \mathsf{Says}\ B\ M\ mb \in tr;$
    $\forall ev.ev \in tr \longrightarrow (\exists\ A'\ B'\ m.\ ev = \mathsf{Says}\ A'\ B'\ m)]\!]$
    $\Longrightarrow \mathsf{knows}\ \mathsf{Spy}\ tr = \mathsf{knows}\ \mathsf{Spy}\ (\mathsf{swap}\ ma\ mb\ tr)$
12. $[\![(\mathsf{noncesOf}\ ma) \cap (\mathsf{used}\ tr) = \emptyset;\ (\mathsf{noncesOf}\ ma) \cap (\mathsf{noncesOf}\ mb) = \emptyset;$
    $\mathsf{nonceDisj}\ mb\ tr;\ \mathsf{noncesOf}\ ma \neq \emptyset]\!]$
    $\Longrightarrow (\mathsf{noncesOf}\ mb) \cap (\mathsf{used}\ (\mathsf{swap}\ ma\ mb\ tr)) = \emptyset$
13. $[\![(\mathsf{noncesOf}\ m) \cap (\mathsf{used}\ tr) = \emptyset;\ (\mathsf{noncesOf}\ m) \cap (\mathsf{noncesOf}\ mb) = \emptyset;$
    $(\mathsf{noncesOf}\ m) \cap (\mathsf{noncesOf}\ ma) = \emptyset]\!]$
    $\Longrightarrow (\mathsf{noncesOf}\ m) \cap (\mathsf{used}\ (\mathsf{swap}\ ma\ mb\ tr)) = \emptyset$

Let $tr' = \mathsf{swap}\ ma\ mb\ tr$. In Lemma 25, part 1 says that the message of the event tr_i is almost the same as that of tr'_i except the case when the message is ma or mb. If the message sent in tr_i is ma, then the counterpart in tr'_i is mb, and vice versa. Part 2 says that each sender and receiver of each event tr_i is the same as those of tr'_i. Part 3 shows that $\mathsf{swap}\ ma\ mb\ tr$ has the same length as tr. Part 4 says that messages observed from tr is the same as those of $\mathsf{swap}\ ma\ mb\ tr$. Part 5 shows that the trace $\mathsf{swap}\ ma\ mb\ tr$ is the same as $\mathsf{swap}\ ma\ mb\ tr$. Part 6, part 7, part 8, and part 9 show some correspondence of an event occurring in tr and the corresponding one in tr'. Part 10 and part 11 show that if $\mathsf{Says}\ A\ M\ ma \in tr$, and $\mathsf{Says}\ B\ M\ mb \in tr$, for Spy, the set of messages and knowledge obtained from tr is the same as those from $\mathsf{swap}\ ma\ mb\ tr$. Part 12 says that nonces of mb will be disjoint from those of $\mathsf{used}\ tr'$ if nonces of ma are disjoint from those of mb, nonces of ma are disjoint

from used tr, nonceDisj mb tr, and nonces of ma are not empty. Part 13 says that nonces of m will be disjoint from those of used tr' if nonces of m are disjoint from those of ma, nonces of m are disjoint from those of mb, and nonces of ma are disjoint from used tr.

8.4.2. swap ma mb tr *is an Onion Routing trace*

Next predicate nonceDisjUntil ma tr says that nonces of ma are disjoint with any other message occurring in any tr' such that tr' is a prefix of trace tr with length of $tr' \leq i$.

```
definition nonceDisjUntil::"msg ⇒ trace ⇒ nat⇒ bool"
where "nonceDisjUntil ma tr i ≡∀ tr'.
(length tr' ≤i∧ tr'∈ tails tr ⟶nonceDisj ma tr')"
```

For a trace $tr \in$ oneOnionSession i M and an event Says A M m occurring tr, we have nonceDisjUntil m tr.

Lemma 26. $[\![tr \in$ oneOnionSession i M; Says A M $m \in$ set tr $]\!] \Longrightarrow$ nonceDisjUntil m tr

The following predicate isRouterRecvMsg m M specifies that m is a message sent to the router M. In the context of this subsection, when we mention ma and mb (see lemmas below), we always mean $ma(mb)$ satisfies isRouterRecvMsg $ma(mb)$ M.

```
definition isRouterRecvMsg::  "msg ⇒agent⇒ bool"
where "isRouterRecvMsg m M≡
(∃ n0 n Y.Y≠M ∧
m= (Crypt (pubEK M) {|Nonce n0,Agent Y,Crypt (pubEK Y) (Nonce n)|})) ∨
(∃ n.  m=(Crypt (pubEK M) (Nonce n)))"
```

The next predicate bothContained specifies that both ma and mb are contained in the messages of tr if the length of $tr \geq i$.

```
definition bothContained::"trace ⇒ msg ⇒ msg ⇒ nat ⇒ agent ⇒ bool"
where "bothContained tr ma mb i M ≡
length tr ≥ i ⟶
((∃ X . Says X M ma ∈ set tr) ∧ (∃ X. Says X M mb ∈ set tr))"
```

Next lemma specifies an invariant on a trace tr in oneOnionSession i M, if both ma and mb are messages sent to the router M, nonces of ma and mb are disjoint, nonces of $ma(mb)$ are disjoint with those any other message in any prefix tr' of tr whose length is less than or equal to i, both ma and mb are contained in the messages of tr is the length of $tr \geq i$, then swap ma mb TR is also a trace in oneOnionSession i M.

Lemma 27. $[\![tr \in$ oneOnionSession i M; ((noncesOf ma)$\cap$(noncesOf mb) $= \emptyset$); nonceDisjUntil ma tr i; nonceDisjUntil mb tr i; bothContained tr ma mb i M; isRouterRecvMsg ma M;isRouterRecvMsg mb $M$$]\!] \Longrightarrow$
(swap ma mb $tr \in$ oneOnionSession i M)

Lemma 27 is rather complex, we must consider three cases: (1) neither ma nor mb occurs in trace tr; (2) only one message $ma(mb)$ occurs in tr, while the other message $mb(ma)$ does not occur in tr; (3) both ma and mb occur in tr. Lemma 27 specifies an invariance which holds in each one of the above three cases. Based on Lemma 26 and Lemma 27, we can conclude an important result: for a trace $tr \in \mathsf{oneOnionSession}\ i\ M$, both ma and mb are sent to the router M by some agents in tr, then $\mathsf{swap}\ ma\ mb\ tr$ is still in $\mathsf{oneOnionSession}\ i\ M$. The proof is by induction on tr, and heavily rely on parts of the Lemma 25.

Theorem 28. $[\![tr \in \mathsf{oneOnionSession}\ i\ M; \mathsf{Says}\ A\ M\ ma \in tr; \mathsf{Says}\ B\ M\ mb \in tr]\!] \Longrightarrow \mathsf{swap}\ ma\ mb\ tr \in \mathsf{oneOnionSession}\ i\ M$

Proof. From the premises that $\mathsf{Says}\ A\ M\ ma \in tr$ and $\mathsf{Says}\ B\ M\ mb \in tr$, it is trivial to prove that the predicate $\mathsf{bothContained}\ tr\ ma\ mb\ i\ M$. We have that they are both messages sent to the router M. Thus the messages ma and mb satisfy that $\mathsf{isRouterRecvMsg}\ ma\ M$ and $\mathsf{isRouterRecvMsg}\ mb\ M$. By Lemma 26, we have $\mathsf{nonceDisjUntil}\ ma\ tr\ i$ and $\mathsf{nonceDisjUntil}\ mb\ tr\ i$. By Lemma 27, we conclude that $\mathsf{swap}\ ma\ mb\ tr \in \mathsf{oneOnionSession}\ i\ M$.

8.4.3. Alignment properties

By Lemma 22, we can show that the relation, composed of two messages sequences of message parts of tr and $\mathsf{swap}\ ma\ mb\ tr$, is $\mathsf{single\_valued}$.

Lemma 29. $[\![tr \in (\mathsf{oneOnionSession}\ i\ M);$
$r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)(\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)))]\!]$
$\Longrightarrow \mathsf{single\_valued}\ r$

Let $r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)(\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)))$, $Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows}\ \mathsf{Spy}\ tr))$;, after applying analyzing operations pairwise on tr, we obtain a relation $\mathsf{analz\_pairs}\ tr\ Kn$. Based on Lemma 29, we show $\mathsf{analz\_pairs}\ r\ Kn$ is single_valued.

Lemma 30. $[\![r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)(\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr));$
$Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows}\ \mathsf{Spy}\ tr)); r' = \mathsf{analz\_pairs}\ r\ Kn;$
$M \notin \mathsf{bad}; tr \in \mathsf{oneOnionSession}\ i\ M;$
$\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr;$
$(m, m') \in r'; (m, m'') \in r']\!]$
$\Longrightarrow m' = m''$

From Lemma 30, we derive a sufficient condition, which is depicted in Lemma 9, in order to prove that $\mathsf{synth\_pairs}\ (\mathsf{analz\_pairs}\ r\ Kn)$ is single_valued.

Lemma 31. $[\![r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)(\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr));$
$Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows}\ \mathsf{Spy}\ tr)); r' = \mathsf{analz\_pairs}\ r\ Kn;$
$M \notin \mathsf{bad}; tr \in \mathsf{oneOnionSession}\ i\ M;$
$\mathsf{nonLeakMsg}\ ma\ M; \mathsf{nonLeakMsg}\ mb\ M;$
$\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr;$
$(m, m') \in r'; (m, m'') \in \mathsf{synth\_pairs}\ r'\ Kn]\!]$
$\Longrightarrow m' = m''$

Notice that two conditions nonLeakMsg ma M and nonLeakMsg mbM must be added in Lemma 31. Without the two conditions, ma(or mb) can be synthesized from some Nonce n if $ma = \{\!|\mathsf{Nonce}\ n, \mathsf{Agent\ Spy}, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK\ Spy}}|\!\}_{\mathsf{pubK}\ M}$. Thus both (ma, ma) and (ma, mb) occur in synth_pairs r' Kn. From Lemma 30, by Lemma 9, we can conclude that synth_pairs (analz_pairs r Kn) is single_valued.

Lemma 32. $[\![r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map\ msgPart}\ tr)(\mathsf{map\ msgPart}\ (\mathsf{swap}\ ma\ mb\ tr))$;
$Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows\ Spy}\ tr))$; $r' = \mathsf{analz\_pairs}\ r\ Kn$;
nonLeakMsg ma M;nonLeakMsg mbM;
$M \notin \mathsf{bad}$; $tr \in \mathsf{oneOnionSession}\ i\ M$;
$\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr$; $\mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr$;
$(m, m') \in \mathsf{synth\_pairs}\ r'\ Kn$; $(m, m'') \in \mathsf{synth_pairs}\ r'\ Kn$$]\!]$
$\Longrightarrow m' = m''$

Because the corresponding relation between tr and swap ma mb tr can guarantee that $r = r^{-1}$, and the reflexivity can be kept by the analz_pairs and synth_pairs operators, then $(\mathsf{synth\_pairs}\ r'\ Kn)^{-1}$ is also single_valued.

Lemma 33. $[\![r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map\ msgPart}\ tr)(\mathsf{map\ msgPart}\ (\mathsf{swap}\ ma\ mb\ tr))$;
$Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows\ Spy}\ tr))$; $r' = \mathsf{analz\_pairs}\ r\ Kn$;
nonLeakMsg ma M;nonLeakMsg mbM;
$M \notin \mathsf{bad}$; $tr \in \mathsf{oneOnionSession}\ i\ M$;
$\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr$; $\mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr$;
$(m, m') \in (\mathsf{synth\_pairs}\ r'\ Kn)^{-1}$; $(m, m'') \in (\mathsf{synth_pairs}\ r'\ Kn)^{-1}$$]\!]$
$\Longrightarrow m' = m''$

8.4.4. Observation equivalence between tr and swap ma mb tr

Let r = zip (map msgPart tr) (map msgPart (swap ma mb tr) and Kn = synth (analz ((knows Spy tr)). For a pair $(ma, mb) \in r$, ma and mb are observation equivalent to each other w.r.t.t knowledge Kn.

Lemma 34. $[\![tr \in \mathsf{oneOnionSession}\ i\ M$;$\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr$;
$\mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr$; $A \notin \mathsf{bad}$; $B \notin \mathsf{bad}$; $M \notin bad$;
$ma = \mathsf{Crypt}\ (\mathsf{pubEK}\ M)\ \{\!|\mathsf{Nonce}\ na_0, \mathsf{Agent}\ Y, \mathsf{Crypt}\ (\mathsf{pubEK}\ Y)\ (\mathsf{Nonce}\ na)|\!\}$;
nonLeakMsg ma M;nonLeakMsg mb $M$$]\!]$
$\Longrightarrow \mathsf{msgEq}\ (\mathsf{synth}\ (\mathsf{analz}\ ((\mathsf{knows\ Spy}\ tr)))\ ma\ mb$

Notice that conditions nonLeakMsg ma M and nonLeakMsg mb M guarantee the correctness of na_0 and some nonce part of mb, which in turn guarantees the observational equivalence between ma and mb.

Next we show that (swap ma mb tr)) is observational equivalent to tr for a spy if tr satisfies some constraints.

If $ma = \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$, ma is sent to the router M by an honest agent A, and mb is also sent to the router M by an honest agent B, then tr is observationally equivalent to swap ma mb tr in the view of the Spy.

Lemma 35. $[\![tr \in \mathsf{oneOnionSession}\ i\ M;$
$ma = \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M};$
$\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr;$
$A \notin \mathsf{bad}; M \notin \mathsf{bad}; B \notin \mathsf{bad};$
$\mathsf{nonLeakMsg}\ ma\ M; \mathsf{nonLeakMsg}\ mb\ M]\!]$
$\Longrightarrow \mathsf{obsEquiv}\ \mathsf{Spy}\ tr\ (\mathsf{swap}\ ma\ mb\ tr)$

Proof. By the definition of view, we can have (a) $\mathsf{view}\ \mathsf{Spy}\ tr = tr$ from $trin\mathsf{oneOnionSession}\ i\ M$. Unfolding the definition of obsEquiv, by part 3 in Lemma 25, we can prove (b) $\mathsf{length}\ (\mathsf{swap}\ ma\ mb\ tr) = \mathsf{length}\ tr$; by part 2 in Lemma 25, we also have (c) $\mathsf{sendRecvMatchL}\ tr\ (\mathsf{swap}\ ma\ mb\ tr$; by part 11 in Lemma 25, we have (d) $\mathsf{set}\ (\mathsf{map}\ \mathsf{msgPart}\ tr) = \mathsf{set}\ ((\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)))$. Let $r = (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)\ (\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)))$ and $Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows}\ Spy\ tr))$, we need to prove (e)$\forall\ m\ m'.(m, m') \in r \longrightarrow \mathsf{msgEq}\ Kn\ m\ m'$. We only need to fix two messages m_1 and m_2 such that $(m_1, m_2) \in \mathsf{set}\ r$, then prove that $\mathsf{msgEq}\ Kn\ m_1\ m_2$. By Lemma 1, we have either (1) $m_1 = m_2$, (2)$m_1 = ma$ and $m_2 = mb$, or (3) $m_1 = mb$ and $m_2 = ma$. For the first case, by Lemma 3, we have $\mathsf{msgEq}\ Kn\ m_1\ m_2$; for case (2) and (3), they can be directly proved by Lemma 34. Let $r' = \mathsf{synth\_pairs}(\mathsf{analz\_pairs}\ r\ Kn)$, by Lemma 32 and 33, we have (f) $\mathsf{single\_valued}\ r'$ and $\mathsf{single\_valued}\ (r')^{-1}$.

From (a)(b)(c)(d)(e)(f), we conclude $\mathsf{obsEquiv}\ \mathsf{Spy}\ tr\ (\mathsf{swap}\ ma\ mb\ tr)$.

8.5. Proving anonymity properties

Let us give two preliminary definitions: the senders in a trace is defined as $\mathsf{senders}\ tr\ M \equiv \{A.\exists m.\mathsf{Says}\ A\ M\ m \in \mathsf{set}\ tr\}$, and a predicate $\mathsf{nonLeakTrace}\ tr\ M \equiv \forall A\ n_0\ n\ Y.\mathsf{Says}\ A\ M\ m \in \mathsf{set}\ tr \longrightarrow A \notin \mathsf{bad} \longrightarrow \mathsf{nonLeakMsg}\ m\ tr$ specifying that tr is a trace where each honest agent sends a message which satisfies $\mathsf{nonLeakMsg}\ m\ tr$.

Message ma' is forwarded to B by the router M, and is originated by some honest agent, and the trace satisfies $\mathsf{nonLeakMsg}\ m\ tr$, then Spy cannot be sure of the honest agent who originates ma' if Spy is an observer. Namely, the sender anonymity holds for the intruder w.r.t. the honest agents who send messages to M in the session modeled by tr.

Theorem 36. $[\![tr \in \mathsf{oneOnionSession}\ i\ M;\ ma' = \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y};$
$\mathsf{Says}\ M\ B\ ma' \in \mathsf{set}\ tr; \mathsf{regularOrig}\ ma'\ tr;\ M \notin \mathsf{bad}; \mathsf{nonLeakTrace}\ tr\ M]\!]$
$\Longrightarrow \mathsf{senderAnomity}\ (\mathsf{senders}\ tr\ M - \mathsf{bad})\ \mathsf{Spy}\ ma'\ tr\ (\mathsf{oneOnionSession}\ i\ M),$

Proof. By unfolding the definition of the predicate senderAnomity, for any agent $X \in (\mathsf{senders}\ tr\ M - \mathsf{bad})$, fix an agent X, we need to construct a trace tr' such that $tr' \in \mathsf{oneOnionSession}\ i\ M$ and $\mathsf{obsEquiv}\ Spy\ tr\ tr'$ and $\mathsf{originates}\ X\ ma\ tr'$. From $\mathsf{Says}\ M\ B\ ma' \in \mathsf{set}\ tr$, by Lemma 17, there exists A and n_0, such that $\mathsf{Says}\ A\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M} \in \mathsf{set}\ tr$. By Lemma 18, we have $\mathsf{originates}\ A\ ma'\ tr$. Obviously, by the fact $\mathsf{regularOrig}\ ma'\ tr$, we have $A \notin \mathsf{bad}$. From the fact $X \in (\mathsf{senders}\ tr\ M - \mathsf{bad})$,

by the definition of senders, there exists an event Says X M $mb \in$ set tr, $X \neq M$, $X \notin$ bad. Let $ma = \{\!|\text{Nonce } n_0, \text{Agent } Y, \{\!|\text{Nonce } n|\}_{\text{pubK } Y}|\}_{\text{pubK } M}$. By nonLeakTrace tr M n, we have both nonLeakMsg ma M and nonLeakMsg mb M. Let $tr' =$ swap ma mb tr, by Lemma 35, we have obsEquiv Spy tr (swap ma mb tr). By Lemma 28, we have swap ma mb $tr \in$ oneOnionSession i M. From the fact Says X M $mb \in$ set tr, by part 6 in Lemma 25, we have Says X M $ma \in$ swap ma mb tr. By Lemma 18, we have originates X ma' (swap ma mb tr). ■

The last result is about the linkability of a sender A and a peeled onion ma. Suppose that an honest agent A sends a message m to the router M, and an agent B receives a message ma from M, the intruder cannot link the message ma' with the agent A provided that there exists at least one agent X who is not A and sends a message to M.

Theorem 37. $[\![tr \in$ oneOnionSession i M; $ma' = \{\!|\text{Nonce } n|\}_{\text{pubK } Y}$;
Says M B $ma' \in$ set tr; regularOrig ma' tr;
Says A M $m' \in$ set tr; $A \notin$ bad; $M \notin$ bad;
$\exists X, mx.$Says X M $mx \in$ set $tr \wedge X \neq A \wedge X \notin$ bad; nonLeakTrace tr $M]\!]$
$\Longrightarrow$ let $AS =$ senders tr $M -$ bad in
unlinkability AS A m tr (oneOnionSession i M)

Proof. Let $runs =$ oneOnionSession i M, $AS =$ senders tr $M -$ bad. By unfolding the definition of the predicate unlinkability, we only need to prove that (1) $tr \vDash \Diamond$Spy $runs$ ($\neg$originates A ma' tr) and (2)senderAnomity AS Spy ma' tr $runs$. Here (1) is our main goal, and (2) is proved in Lemma 36.

From the premise, there exist X and mx such that Says X M $mx \in$ set tr, $X \neq A$, and $X \notin bad$. From Says M B $ma' \in$ set tr, by Lemma 17, there exists a message m, an agent A', a nonce n_0, such that ma has the form of

$$\text{Says } A' \ M \ \{\!|\text{Nonce } n_0, \text{Agent } Y, \{\!|\text{Nonce } n|\}_{\text{pubK } Y}|\}_{\text{pubK } M} \in \text{set } tr.$$

Obviously, by the fact regularOrig ma' tr, we have $A' \notin$ bad.In order to prove (1), by unfolding the definition of the diamond operator, we only need construct a trace tr' such that obsEquiv Spy tr tr' and $\neg$ $originates$ A ma' tr. Here we do case analysis on A'.

If $A' \neq A$, then (1) can be proved immediately. Obviously obsEquiv Spy tr tr, $tr \in$ oneOnionSession i M. By Lemma 20, we have $\neg$originates A ma' tr. Otherwise, from $A' = A$, we have $X \neq A'$. let $tr' =$ swap ma mx tr, by Lemma 35, we have obsEquiv Spy tr tr'. By Lemma 28, we have $tr' \in$ oneOnionSession i M. From Says X M $mx \in$ set tr and Says A M $ma \in$ set tr, by Lemma 6, we have Says X M $ma \in$ set tr' and Says A M $mx \in$ set tr'. From $X \neq A$, by Lemma 20, immediately we have $\neg$originates A ma' tr'. ■

8.6. A weakness of the protocol

Here, we show a weakness of the onion routing protocol, which is hinted by the premise cond tr M n. Namely, without this condition, the sender anonymity and unlinkability may not hold. For example, consider the session shown in

Fig. 1, the trace tr in (1) is not observationally equivalent to that in (2) when $C = D = \mathsf{Spy}$, $na = na'$, $nb = nb'$, and $na \neq nb$. Because after the router M forwards messages $\{\!|\mathsf{Nonce}\ na|\!\}_{\mathsf{pubK}\ \mathsf{Spy}}$ and $\{\!|\mathsf{Nonce}\ nb|\!\}_{\mathsf{pubK}\ \mathsf{Spy}}$, the Spy can analyse na and nb respectively, and distinguish the two nonces, then he can distinguish the two messages $\{\!|\mathsf{Nonce}\ na, \mathsf{Agent}\ \mathsf{Spy}, \{\!|\mathsf{Nonce}\ na|\!\}_{\mathsf{pubK}\ \mathsf{Spy}}|\!\}_{\mathsf{pubK}\ M}$ and $\{\!|\mathsf{Nonce}\ nb, \mathsf{Agent}\ \mathsf{Spy}, \{\!|\mathsf{Nonce}\ nb|\!\}_{\mathsf{pubK}\ \mathsf{Spy}}|\!\}_{\mathsf{pubK}\ M}$ at last.

9. Conclusion and Future Work

We have formalized the notion of provable anonymity in the theorem prover Isabelle/HOL. We propose an inductive definition of message distinguishability based on the observer's knowledge, then define message equivalence as the negation of message distinguishability. Next, we define observational equivalence of two traces using the message equivalence, and define the semantics of anonymity properties in an epistemic logical framework. In the end, we inductively formalize the semantics of Crowds and Onion Routing, and formally prove anonymity properties for the protocols in our formal framework, i.e., sender anonymity for Crowds, sender anonymity and unlikability for Onion Routing.

When we prove that anonymity properties, e.g., sender anonymity, hold for a trace under consideration, we need to consider the existence of another trace which is observationally equivalent to the given trace, but differs, for example, in the sender of some message. This is the essence of information hiding on the senders or the linkage between a message and its sender, which makes the analysis of anonymity different from analysis on secrecy and authentication. For secrecy and authentication, normally the focus is on individual traces. However, the observer decides whether two traces are observationally equivalent according to his knowledge obtained in two traces, which usually boils down to the secrecy of some terms. Therefore, the induction proof method used in the analysis of secrecy properties can still be applied here.

In future, we plan to apply our framework to more case studies. We would like to check whether our framework can be easily generalized to model other different kinds of privacy and information hiding properties and to model protocols that allow more cryptographic primitives. Theoretically, we believe the inductive approach proposed in this paper can be extended because only additional induction rules are required. In particular, it is interesting for us to find out whether the method of constructing an observationally equivalent trace using the swap function is generally applicable. In the literature, simulation-based proof techniques similar to the our swap function have been proposed [17, 18]. Trace anonymity [27] is formalized using I/O automaton and the Larch prover is employed for check trace anonymity [17]. An anonymous fair exchange e-commerce protocol that is claimed to satisfy customer's anonymity is analyzed using the OTS/CafeOBJ method [18] following the approach proposed in [17]. However, both approaches only consider a weaker intruder, which does not have the same ability to distinguish messages as we presented in this paper.

References

[1] A. Baskar, R. Ramanujam, and S. P. Suresh. Knowledge-based modelling of voting protocols. In *Proc. 11th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 62–71. ACM, 2007.

[2] M. Bhargava and C. Palamidessi. Probabilistic anonymity. In *Proc. 16th Conference on Concurrency Theory*, volume 3653 of *LNCS*, pages 171–185. Springer, 2005.

[3] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[4] T. Chothia. Analysing the mute anonymous file-sharing system using the pi-calculus. In *Proc. 26th Conference on Formal Methods for Networked and Distributed Systems*, volume 4229 of *LNCS*, pages 115–130, 2006.

[5] T. Chothia, S. M. Orzan, J. Pang, and M. Torabi Dashti. A framework for automatically checking anonymity with μCRL. In *Proc. 2nd Symposium on Trustworthy Global Computing*, volume 4661 of *LNCS*, pages 301–318. Springer, 2007.

[6] S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.

[7] Y. Deng, C. Palamidessi, and J. Pang. Weak probabilistic anonymity. In *Proc. 3rd Workshop on Security Issues in Concurrency*, volume 180 of *ENTCS*, pages 55–76, 2007.

[8] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium*, pages 303–320, 2004.

[9] D. Dolev and A. C.-C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[10] N. Dong, H. L. Jonker, and J. Pang. Formal analysis of privacy in an eHealth protocol. In *Proc. 17th European Symposium on Research in Computer Security*, volume 7459 of *LNCS*, pages 325–342. Springer, 2012.

[11] F. D. Garcia, I. Hasuo, W. Pieters, and P. van Rossum. Provable anonymity. In *Proc. 3rd Workshop on Formal Methods in Security Engineering*, pages 63–72. ACM, 2005.

[12] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding routing information. In *Proc. 1st Workshop on Information Hiding*, LNCS 1174, pages 137–150. Springer, 1996.

[13] J. Y. Halpern and K. R. O'Neill. Anonymity and information hiding in multiagent systems. *Journal of Computer Security*, 13(3):483–514, 2005.

[14] D. Hughes and V. Shmatikov. Information hiding, anonymity and privacy: A modular approach. *Journal of Computer Security*, 12(1):3–36, 2004.

[15] H. L. Jonker and E. P. de Vink. Formalising receipt-freeness. In *Proc. 9th Conference on Information Security*, volume 4176 of *LNCS*, pages 476–488. Springer, 2006.

[16] H. L. Jonker, S. Mauw, and J. Pang. A formal framework for quantifying voter-controlled privacy. *Journal of Algorithms in Cognition, Informatics and Logic*, 64(2-3):89–105, 2009.

[17] Y. Kawabe, K. Mano, H. Sakurada, and Y. Tsukada. Theorem-proving anonymity of infinite state systems. *Information Processing Letters*, 101(1):46–51, 2007.

[18] W. Kong, K. Ogata, J. Cheng, and K. Futatsugi. Trace anonymity in the OTS/CafeOBJ method. In *Proc. 8th IEEE International Conference on Computer and Information Technology*, pages 754–759. IEEE, 2008.

[19] S. Kremer and M. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Proc. 14th European Symposium on Programming*, volume 3444 of *LNCS*, pages 186–200. Springer, 2005.

[20] Y. Li and J. Pang. An inductive approach to provable anonymity. In *Proc. 6th Conference on Availability, Reliability and Security*, pages 454–459. IEEE Computer Society, 2011.

[21] L. Luo, X. Cai, J. Pang, and Y. Deng. Analyzing an electronic cash protocol using applied pi-calculus. In *Proc. 5th Conference on Applied Cryptography and Network Security*, volume 4521 of *LNCS*, pages 87–103. Springer, 2007.

[22] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[23] J. Pang and C. Zhang. How to work with honest but curious judges? (preliminary report). In *Proc. 7th Workshop on Security Issues in Concurrency*, volume 7 of *EPTCS*, pages 31–45, 2009.

[24] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[25] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management, April 2010.

[26] M. K. Reiter and A. D. Rubin. Crowds: anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[27] S. Schneider and A. Sidiropoulos. CSP and anonymity. In *Proc. 4th European Symposium on Research in Computer Security*, volume 1146 of *LNCS*, pages 198–218. Springer, 1996.

[28] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3/4):355–377, 2004.

[29] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *Proc. 18th IEEE Symposium on Security and Privacy*, pages 44–54. IEEE, 1997.

[30] L. Yan, K. Sere, X. Zhou, and J. Pang. Towards an integrated architecture for peer-to-peer and ad hoc overlay network applications. In *Proc. 10th Workshop on Future Trends in Distributed Computing Systems*, pages 312–318. IEEE Computer Society, 2004.

Appendix

In the appendix, we briefly present some Isabelle concepts, notations and commands, and our notation conventions for variables in our work.

Isabelle's meta-logic is the intuitionistic fragment of Church's theory of simple types, which can be used to formalize an object-logic which we need [22]. Normally, we use rich infrastructure of the object-logics such as HOL to formalize some theory, which has been provided by Isabelle system. Important connectives of the meta-logic are as follows: implication ($\Longrightarrow$) is for separating premises and conclusion of theorems; equality ($\equiv$) definitions; universal quantifier ($\bigwedge$) parameters in goals. In our work, we use the object-logic HOL to formalize the anonymity theory. Therefore, we briefly show how to use HOL to formalize a theory.

Theories. Working with Isabelle means creating theories. A theory is a file with a named collection of types, functions, and theorems, proofs. The general format of a theory T is as follows:

theory $T = B_1 + B_2 + \ldots + B_n$;
declarations for types, *definitions*, *lemmas*, *and proofs*
end

where $B_1, B_2, \ldots, B_n$ are are the names of existing theories that T is based on. In our case, we only need to import HOL library `Main` to create our theory `anonymity.thy`.

Types. There are basic types such as bool, the type of truth values; nat, the type of natural numbers. Function types are denoted by $\Rightarrow$, and product types by $\times$. Types can also be constructed by type constructors such as list and set. For instance, nat list declares the type of lists whose members are natural numbers.

Terms. Forms of terms used in this paper are rather simple. It is simply a constant or variable identifier, or a function application such as $f\ t$, where f is a function of type $\tau_1 \Rightarrow \tau_2$, and t is a term of type τ_1. Formulas are terms of type bool. bool has two basic constants True and False and the usual logical connectives (in decreasing order of priority): $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\forall$, and $\exists$, all of which (except the unary $\neg$) associate to the right. Note that the logical connectives introduced here are used in the object-logic HOL.

Introducing new types. There are three kinds of commands for introducing new types. typedecl *name* introduces new "opaque" type name without definition; types $name = \tau$ introduces an abbreviation name for type τ. datatype command can introduce a recursive data type. A general datatype definition is of the form

$$\mathsf{datatype}\ (\alpha_1, \ldots, \alpha_n) = C_1\ \tau_{11}\ \ldots\ \tau_{1k_1}\ \mid \ldots \mid\ C_m\ \tau_{m1} \ldots \tau_{mk_m}$$

where α_i are distinct type variables (the parameters), C_i are distinct constructor names and τ_{ij} are types. Note that n can be 0, i.e., there is no type parameters in datatype declaration.

Definition commands. consts command declares a function's name and type. defs gives the definition of a declared function. constdefs combines the effect of consts and defs. Combining a consts and inductive commands, we can give an inductive definition for a set. An inductively defined set S is typically of the following form:

consts $S :: \tau$set inductive S intros
$rule_1 : [\![a_{11} \in S; \ldots; a_{1k_1} \in S; A_{11}, \ldots, A_{1i_1}]\!] \Longrightarrow a_1 \in S$
...
$rule_n : [\![a_{n1} \in S; \ldots; a_{nk_n} \in S; A_{n1}, \ldots, A_{ni_n}]\!] \Longrightarrow a_n \in S$

Lemmas. In Isabelle's traditional style, we use the notation lemma *name* : $[\![A_1; A_2; \ldots; A_n]\!] \Longrightarrow B$ to donote that with assumptions $A_1, \ldots, A_n$, we can derive a conclusion B. In Isar's style, a lemma is written as lemma *name* : assumes a_1 : "A_1" and ... and a_n : "A_n" shows B.

Notation conventions. Throughout this paper, we use the conventions for meta-variables as follows:

i, j	range over natural numbers for lengths of traces
m, m', x, y	range over messages
n, n', n_1, n_2	ranges over nonces
r, r'	ranges over sets of message pairs
A, B, M, X, Y, R	range over agent names
k, k_1, k_2	range over keys
ev, ev_1, ev_2	range over events
tr, tr_1, tr_2	range over traces
G, H, Kn, $Know$	range over message sets