An Inductive Approach to Provable Anonymity

Yongjian Li
The State Key Laboratory of Computer Sciences
Institute of Software
Chinese Academy of Sciences

Jun Pang
Computer Science and Communications
University of Luxembourg

***Abstract*—We formalise in a theorem prover the notion of provable anonymity proposed by Garcia et al. Our formalization relies on inductive definitions of message distinguish ability and observational equivalence over observed traces by the intruder. Our theory differs from its original proposal which essentially boils down to the existence of a reinterpretation function. We build our theory in Isabelle/HOL to have a mechanical framework for the analysis of anonymity protocols. Its feasibility is illustrated through the onion routing protocol.**

I. 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 has 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 might involve sensitive personal data, like electronic auctions, voting, anonymous broadcasts, file-sharing etc. For example, users want to keep anonymous when they visit a particular web site or post their political opinions on a public bulletin board.

Due to its subtle nature, anonymity has been the subject of many theoretical studies and formal verification [1], [2], [3], [4], [5]. The proposed definitions aim to capture different aspects of anonymity (either possibilisticor probabilisticand formal verification treats systems in different application domains, such as electronic voting systems electronic cash protocols file sharing. However, automatic approaches to the formal verification of anonymity have mostly focused on the model checking approach on systems with fixed configurations [1], while theorem proving is a more suitable approach when dealing with general systems of infinite state spaces. We address this situation by investigating the possibility of using a powerful general-purpose theorem prover, Isabelle/HOL [6], to semi-automatically verify anonymity properties.

We start by formalising the notion of provable anonymity proposed by Garcia et al. [2]. Their key idea is to define observational equivalence between protocol traces. Two traces are to be considered equivalent if an intruder cannot distinguish them, i.e., he cannot find any meaningful difference. The distinguishing ability of the intruder is formalised as the ability to distinguish two messages, which is in turn based on message structures and relations between random looking messages. Central to their framework is the *reinterpretation function* proposed by Garcia et al. [2]. 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.

Our contribution: Our formalization of observational equivalence between traces relies on a definition of message distinguishability. Observational equivalence of traces is in the center of the epistemic framework – an agent knows a fact of a certain trace if that fact is true in all traces that are observationally equivalent to that trace. We build our theory in Isabelle/HOL [6] to have a mechanical framework for the analysis of anonymity protocols. We illustrate the feasibility of the mechanical framework through the onion routing protocol [7]. We inductively define the semantics of an onion routing protocol 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 onion trace for a given trace. To the best of our knowledge, theory of anonymity has not been formalised in a theorem prover yet. Our work aims to to bridge this gap. All lemmas in the paper are proved semi-automatically in Isabelle/HOL. Proofs are mostly omitted for the sake of brevity.

II. Preliminaries

A. Agents, messages and events

Agents send or receive messages. There are three kinds of agents: the server, the friendly 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:

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

where A is an element from agents, N from natural numbers, and K from natural numbers. Here we use K^{-1} to denote the inverse key of K. 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.

Two operators parts and analz are inductively defined on a message set H. Their definition is taken from [8] 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.

The parts operator can be used to define the subterm relation $\sqsubset$: $h_1 \sqsubset h_2 \equiv h_1 \in \text{parts}\{h_2\}$. Here K is not regarded 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.

A protocol's behaviour is specified as the set of possible traces of events. A trace model is concrete and easy to explain. An event is of the form: Says A B m, which means that A send 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 formalises 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 $msg \equiv \{m.\exists n.m \sqsubset msg \wedge m =$ Nonce $n\}$ defines the set of nonces occuring 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.

Next we define a set of special lists: mutualDiffL. If $L \in$ mutualDiffL, $i, j <$ length L, and $i \neq j$, then we have $L_i \neq L_j$. Here L_i is the i-th element of the list L.

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

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 $L \in$ mutualDiffL, then single_valued zip L L' for any L'.

B. 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 authentication or secrecy properties 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, we would like to adapt a weaker attacker model. We assume that the intruder is *passive* in the sense that he observes all network traffic, but does not actively modify the messages or inject new messages. He can analyze the messages he has observed, which is modelled by the operator analz. In later section, we will point out that some anonymity properties cannot be kept if we have the Dolev-Yao intruder model.

III. Message Distinguishability

In this section, we focus on modelling the ability for the 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 $m = \{\!|g|\!\}_{k_1}$ and $n = \{\!|h|\!\}_{k_2}$, then the agent must use the knowledge $Know$ he possesses to decide whether the two messages are different. There are cases as shown below:

- Both k_1 and k_2 are in $Know$, g, h are in $Know$ as well, and the agent can distinguished g and h, then he also can tell the difference between m and m' as he knows that m and m' are different encrypted messages;

- If one of $\{k_1^{-1}, k_2^{-1}\}$ is in $Know$, and $k_1^{-1} \neq k_2^{-1}$, then the agent can also tell the difference between them as he knows that the two messages can be decrypted using different keys;
- If $k_1^{-1}, k_2^{-1} \in Know$, and the agent can distinguish g and h, then he also can tell the difference between m and m' as he knows that m and m' can be decrypted into different submessages by using k_1^{-1};
- In case when 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.

```
constdefs basicDiff:: "msg⇒msg⇒bool"
"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 M:: "msg set" where
            basic: "⟦x∈M; y∈M; basicDiff x y⟧
            ⟹ (x,y)∈ Diff M"
            | MPLDiff: "⟦w∈M; z∈M; (x,y)∈Diff M⟧
            ⟹ (MPair x w, MPair y z)∈Diff M"
            | MPRDiff: "⟦w∈M; z∈M; (x,y)∈Diff M⟧
            ⟹ (MPair w x, MPair z y)∈Diff M"
            | CryptDiff: "⟦(Key k1∈M); (Key k2∈M);
              (x,y)∈Diff M⟧
            ⟹ (Crypt k1 x, Crypt k2 y)∈Diff M"
            | DeCryptDiff: "⟦(Key (invKey k1)∈M);
              (Key (invKey k2)∈M);
            (x,y)∈Diff M⟧
            ⟹ (Crypt k1 x, Crypt k2 y)∈Diff M"
            | DecryptDiff1: "⟦(Crypt k1 x1)∈M;
              (Crypt k2 x2)∈M; (Key (invKey k1))∈M;
              (Key (invKey k1))≠(Key (invKey k2))⟧
            ⟹ (Crypt k1 x1, Crypt k2 x2): Diff M"
            | DecryptDiff2: "⟦(Crypt k1 x1)∈M;
              (Crypt k2 x2)∈M; (Key (invKey k2))∈M;
              (Key (invKey k1))≠(Key (invKey k2))⟧
            ⟹ (Crypt k1 x1, Crypt k2 x2): Diff M"
```

IV. Observational Equivalence

We first 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 Knows m1 m2 ≡ (m1, m2)∉ Diff Knows"
```

Obviously, observational equivalence between messages w.r.t. a knowledge set $Know$ is reflexive, symmetric.

Intuitively, we can lift observational equivalence to traces: two sequences of messages in two traces look the same to an agent if they are the same for the messages the agent understands and if a message in one sequence is observationally equivalent to the corresponding message in the other sequence w.r.t. the knowledge which the agent has obtained from the two traces. Besides the requirement of message matching, we also require that the sender and receiver of an event in a trace is the same as those of the corresponding event in the other sequence. For events ev_1 and ev_2, we define $\mathsf{SRMatch}\ ev_1\ ev_2 \equiv \mathsf{sender}\ ev_1 = \mathsf{sender}\ ev_2 \wedge \mathsf{receiver}\ ev_1 = \mathsf{receiver}\ ev_2$. For two traces tr and tr', $\mathsf{SRMatchL}\ tr\ tr' \equiv \mathsf{length}\ tr = \mathsf{length}\ tr' \wedge \forall\ i.i < \mathsf{length}\ tr \longrightarrow \mathsf{SRMatch}\ tr_i\ tr'_i$. The predicate $\mathsf{SRMatchL}\ tr\ tr'$ 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. We extend observational equivalence to traces, written as $tr \approx_A tr'$:

- tr and tr'_i have the same length;
- For events tr_i and tr'_i, the senders and receivers of tr_i are the same as those of tr'_i.
- Furthermore, $\mathsf{msgPart}\ tr_i$ and $\mathsf{msgPart}\ tr'_i$ are observationally equivalent to each other w.r.t. the knowledge obtained after observing the two traces.
- At last single_valued H and single_valued H^{-1} guarantee that an agent cannot reinterpret an event differently.

```
constdefs obsEquiv::"agent⇒trace⇒trace⇒bool"
      "obsEquiv A tr tr′≡ length tr=length tr′∧
          SRMatchL tr tr′∧
          (let H=set (zip (map msgPart tr)
                  (map msgPart tr′)) in
           let Kn=analz ((knows A tr) ∪
                  (knows A tr′)) in
          (∀x y. (x,y)∈ H ⟶ msgEq Kn x y)
           ∧ single_valued H ∧ single_valued H⁻¹)"
```

V. Epistemic Operators and Anonymity Properties

Using the observational equivalence relations over a trace set of possible worlds, we can formally introduce epistemic operators [2] as follows:

```
constdefs box::"agent⇒trace⇒trace set⇒
          assertOfTrace⇒bool"
      "box A r rs Assert≡
          ∀r′.r′∈rs⟶obsEquiv A r r′ ⟶(Assert r′)"

constdefs diamond::"agent⇒trace⇒trace set⇒
          assertOfTrace⇒bool"
      "diamond A r rs Assert≡
          ∃r′.r′∈rs ∧obsEquiv A r r′
          ∧(Assert r′)"
```

For notation convenience, we write $r \models \Box\ A\ rs\ \varphi$ for box $A\ r\ rs\ \varphi$, and $r \models \Diamond\ A\ rs\ \varphi$ for diamond $A\ r\ rs\ \varphi$. Note that φ is a predicate on a trace. Intuitively, $r \models \Box\ A\ rs\ \varphi$ means that for any trace r' in rs, if r' is observationally equivalent to r for agent A, then r' satisfies the assertion φ. On the other hand, $r \models \Diamond\ A\ rs\ \varphi$ menas that there is a trace r' in rs, r' is observationally equivalent to r for agent A and r' 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.

A. Sender anonymity

Suppose that r is a trace of a protocol in which a message m is originated by some agent. We say that r provides sender anonymity with anonymity set AS w.r.t 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 r rs≡ (∀X.X:AS⟶
          r ⊨◇B rs (originates X m))"
```

Here, AS is the set of agents who are under consideration, and rs 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 rs. Therefore, this means that B cannot be sure of anyone who originates this message.

B. Unlinkability

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

```
constdefs unlinkability::"agent set⇒agent⇒msg⇒
          trace⇒trace set⇒bool"
      "unlinkability AS A m r rs≡
          (let P= λX m' r. sends X m' r in
          (¬(r ⊨□ Spy rs (P A m)) ∧
          (∀X.X:AS ⟶r ⊨◇Spy rs (P A m)))"
```

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

VI. Case Study: Onion Routing Protocol

A. Modeling the protocol

In our work, we model a simplified onion routing protocol system, composed of a user set AS and a router M, with $M \notin AS$. We also assume that each agent can send a message before the router M launch a batch of forwarding process, and the router does not accept any message when it is forwarding messages.

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

In this definition, there are four induction rules. Rule `Nil` specifies an empty trace. The other rules specify trace's extension with protocol steps. The ideas behind the other induction rules are illustrated as follows. More precisely,

- If the length of the current trace is less than k, namely, M is still in a receiving status, X (or Y) and M are distinct, and both $n0$ and n are fresh, then 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 k, 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 k, namely, M is in a forwarding status, a message $\{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$ has been received by the router, but the peeled onion $\{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}$ has not been forwarded, 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.

B. Properties on protocol sessions

As mentioned in a previous section, 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.

a) Secrecy properties.: If the router M is honest, B is also honest, and B sends a message $\{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$ to M, either $n_0 \neq n$ or $Y \notin \mathsf{bad}$, then $\mathsf{Nonce}\ n_0$ cannot be analyzed by the intruder.

Lemma 1
$[\![tr \in \mathsf{oneOnionSession}\ k\ M; n_0 \neq n \lor Y \notin \mathsf{bad}; \mathsf{Says}\ B\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M} \in tr; M \notin \mathsf{bad}; B \notin \mathsf{bad}]\!] \Longrightarrow \mathsf{Nonce}\ n_0 \notin \mathsf{analz}\ (\mathsf{spies}\ evs)$

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

Lemma 2
$[\![tr \in \mathsf{oneOnionSession}\ k\ M; \mathsf{Says}\ B\ M\{\!|\mathsf{Nonce}\ n_0|\!\}_{\mathsf{pubK}\ M} \in tr; M \notin \mathsf{bad}; B \notin \mathsf{bad}]\!] \Longrightarrow \mathsf{Nonce}\ n_0 \notin \mathsf{analz}\ (\mathsf{spies}\ evs)$

b) 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 for some nonce n_0 $\{\!|\mathsf{Nonce}\ n_0, Agent\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$.

Lemma 3
$[\![tr \in \mathsf{oneOnionSession}\ k\ M;$
$ma' = \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}; \mathsf{Says}\ M\ B\ ma' \in \mathsf{set}\ tr;]\!] \Longrightarrow$
$\exists n_0\ A.\mathsf{Says}\ A\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M} \in \mathsf{set}\ tr$

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 4
$[\![tr \in \mathsf{oneOnionSession}\ k\ M;$
$ma' = \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}; \mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; ma' \sqsubset ma]\!] \Longrightarrow \mathsf{originates}\ A\ ma'\ tr$

For a trace $tr \in \mathsf{oneOnionSession}\ k\ M$, an agent A sends the router M a message m, then A is not the router M.

Lemma 5
$[\![tr \in \mathsf{oneOnionSession}\ k\ M; \mathsf{Says}\ A\ M\ m \in \mathsf{set}\ tr]\!] \Longrightarrow A \neq M$

c) 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 6
$[\![tr \in \mathsf{oneOnionSession}\ k\ M;$
$\mathsf{Says}\ X\ M\ ma; \mathsf{Says}\ Y\ M\ mb]\!] \Longrightarrow (X = Y \wedge ma = mb) \vee (\mathsf{noncesOf}\ ma) \cap (\mathsf{noncesOf}\ mb) = \emptyset$

From Lemma 6, we can easily derive that once a nonce n occurs in a message sent by an agent X, then another agent Y cannot send a message containing the same nonce n.

Lemma 7
$[\![tr \in \mathsf{oneOnionSession}\ k\ M; \mathsf{Says}\ X\ M\ ma;$
$X \neq Y; \mathsf{Nonce}\ n \sqsubset ma]\!] \Longrightarrow \neg\mathsf{sends}\ Y\ (\mathsf{Nonce}\ n)\ tr$

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 8
$[\![tr \in \mathsf{oneOnionSession}\ k\ M]\!] \Longrightarrow \mathsf{map}\ \mathsf{msgPart}\ tr \in \mathsf{mutualDiffL}$

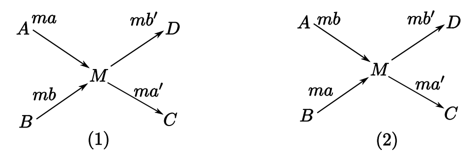
$(\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)\ L)$ must be single_valued if tr is in a trace of the onion routing protocol.

Lemma 9 $[\![tr \in \mathsf{oneOnionSession}\ k\ M]\!]$
$\Longrightarrow \mathsf{single\_valued}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)\ L)$

C. Traces swapping two messages

By definition of sender anonymity, the proof strategy of such property is roughly as follows: fix an agent X, we need to prove the existence of an observationally equivalent trace tr' w.r.t. a given trace tr, where both tr and tr' are some protocol sessions. Obviously, this means a construction of an observationally equivalent trace tr'. In this section, we discuss this construction method in details.

We define a function swap $ma\ mb\ tr$, which returns another trace tr' satisfying that the sender and receiver of any event tr'_i is the same as those in tr_i, but the sent message of tr'_i is swapped as mb if that of tr_i is ma, and as ma if that of tr_i is mb, otherwise it is kept the same as that of tr_i.



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

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

Fig. 1. An illustration of function swap.

```
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 the onion routing protocol, Fig. 1 illustrates the correspondence between tr and the function 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 swap $ma\ mb\ tr$ is formalised as the following lemma.

Lemma 10 *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) $\mathsf{swap}\ ma\ mb\ tr = \mathsf{swap}\ mb\ ma\ tr$
5) $[\![(\mathsf{Says}\ X\ M\ ma \in \mathsf{set}\ tr)]\!]$
$\Longrightarrow \mathsf{Says}\ X\ M\ mb \in \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr))$
6) $[\![(\mathsf{Says}\ X\ M\ mb \in \mathsf{set}\ tr)]\!]$
$\Longrightarrow \mathsf{Says}\ X\ M\ ma \in \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr))$
7) $[\![m \neq ma; m \neq mb; (\mathsf{Says}\ X\ M\ m) \in \mathsf{set}\ tr]\!]$
$\Longrightarrow (\mathsf{Says}\ X\ M\ m \in \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr))$
8) $[\![m \neq ma; m \neq mb; (\mathsf{Says}\ X\ M\ m) \notin \mathsf{set}\ tr]\!]$
$\Longrightarrow (\mathsf{Says}\ X\ M\ m \notin \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr))$
9) $[\![\mathsf{Says}\ A\ M\ ma \in tr; \mathsf{Says}\ B\ M\ mb \in tr; A \neq \mathsf{Spy}; B \neq \mathsf{Spy}]\!]$
$\Longrightarrow \mathsf{knows}\ \mathsf{Spy}\ tr = \mathsf{knows}\ \mathsf{Spy}\ (\mathsf{swap}\ ma\ mb\ tr)$

Based on the lemma 10, we can conclude an important result: for a trace $tr \in \mathsf{oneOnionSession}\ k\ M$, both ma and mb are sent to the router M by some agents in tr, then swap $ma\ mb\ tr$ is still in oneOnionSession $k\ M$.

Theorem 1
$[\![tr \in \mathsf{oneOnionSession}\ k\ M; \mathsf{Says}\ A\ M\ ma \in tr; \mathsf{Says}\ B\ M\ mb \in tr]\!] \Longrightarrow \mathsf{swap}\ ma\ mb\ tr \in \mathsf{oneOnionSession}\ k\ M$

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 11
$[\![tr \in \mathsf{oneOnionSession}\ k\ M;$
$ma = \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M};$
$\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr;$
$A \notin \mathsf{bad}; M \notin \mathsf{bad}; B \notin \mathsf{bad}; n_0 \neq n \vee Y \notin \mathsf{bad}]\!]$
$\Longrightarrow \mathsf{obsEquiv}\ \mathsf{Spy}\ tr\ (\mathsf{swap}\ ma\ mb\ tr)$

D. Proving anonymity properties

Message ma' is forwarded to B by the router M, and is originated by some honest agent, and the nonce n satisfies a constraint cond $tr\ M\ n$, which will be explained in details later, then spy cannot be sure of the honest agent who originates ma'. Namely, the sender anonymity holds for the intruder w.r.t. the honest agents who send messages to M in the session modelled by tr.

Lemma 12
$[\![tr \in \mathsf{oneOnionSession}\ k\ M;$
$ma' = \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y};$
$\mathsf{Says}\ M\ B\ ma' \in \mathsf{set}\ tr; \mathsf{regularOrig}\ ma'\ tr;$
$M \notin \mathsf{bad}; \mathsf{cond}\ tr\ M]\!] \Longrightarrow$
$\mathsf{senderAnomity}\ (\mathsf{senders}\ tr\ M - \mathsf{bad})$
$\mathsf{Spy}\ ma'\ tr\ (\mathsf{oneOnionSession}\ k\ M)$, *where* $\mathsf{senders}\ tr\ M \equiv \{A. \exists m. \mathsf{Says}\ A\ M\ m \in \mathsf{set}\ tr\}$, *and* $\mathsf{cond}\ tr\ M \equiv \forall A\ n_0\ n\ Y. \mathsf{Says}\ A\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M} \in \mathsf{set}\ tr \longrightarrow (Y \notin \mathsf{bad} \vee n_0 \neq n)$

The premise cond $tr\ M\ n$ says that if a nonce n is originated in a message $\{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$ in the trace tr, then either $Y \notin \mathsf{bad}$ or $n_0 \neq n$, this guarantees the secrecy of n.

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.

Lemma 13
$[\![tr \in \mathsf{oneOnionSession}\ k\ M;$
$ma' = \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y};$
$\mathsf{Says}\ M\ B\ ma' \in \mathsf{set}\ tr; \mathsf{regularOrig}\ ma'\ tr;$
$\mathsf{Says}\ A\ M\ m' \in \mathsf{set}\ tr; A \notin \mathsf{bad}; M \notin \mathsf{bad};$
$\exists X, mx. \mathsf{Says}\ X\ M\ mx \in \mathsf{set}\ tr \wedge X \neq A \wedge X \notin \mathsf{bad}; \mathsf{cond}\ tr\ M\ n]\!]$
$\Longrightarrow$ let AS= senders $tr\ M - \mathsf{bad}$ *in*
unlinkability $AS\ A\ m$ (oneOnionSession $k\ M$)

VII. Conclusion and Future Work

In this work, we formalise the notion of provable anonymity in the theorem prover Isabelle/HOL. First we propose an inductive definition of message distinguishability based on the observer's knowledge, then define the 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 formalise the semantics of the onion routing protocols, and formally prove that sender anonymity and unlikability hold for the protocol in Isabelle/HOL.

When we prove that properties such as 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 the 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 used here. This may be the relation between analysis on classical protocol properties on secrecy and that on anonymity properties.

In future, we will apply our framework to more case studies. We would also like to check whether our framework can be easily generalised to model different kinds of privacy and information hiding properties and to model protocols which allow more cryptographic primitives. Theoretically, we believe this inductive approach 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 general enough.

References

[1] S. Schneider and A. Sidiropoulos, "CSP and anonymity," in *Proc. 4th European Symposium on Research in Computer Security*, ser. LNCS, vol. 1146. Springer, 1996, pp. 198–218.

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

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

[4] S. Mauw, J. Verschuren, and E. P. d. Vink, "A formalization of anonymity and onion routing," in *Proc. 9th European Symposium on Research in Computer Security*, ser. LNCS, vol. 3193. Springer, 2004, pp. 109–124.

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

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

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

[8] L. C. Paulson, "The inductive approach to verifying cryptographic protocols," *Journal of Computer Security*, vol. 6, no. 1-2, pp. 85–128, 1998.

[9] D. Dolev and A. C.-C. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.