Under consideration for publication in Formal Aspects of Computing

Formalizing Provable Anonymity in Isabelle/HOL

Yongjian Li
Chinese Academy of Sciences
Institute of Software
State Key Laboratory of Computer Science &
Capital Normal University
College of Information Engineering

Jun Pang
University of Luxembourg
Faculty of Science, Technology and Communication
Computer Science and Communications &
University of Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust

Abstract. We formalize in a theorem prover the notion of provable anonymity. Our formalization relies on inductive definitions of message distinguishing 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 distinguishing messages with respect to a knowledge set for the intruder. We build our theory in Isabelle/HOL to achieve a mechanical framework for the analysis of anonymity protocols. Its feasibility is illustrated through two case studies of the Crowds and Onion Routing protocols.

Keywords: Anonymity, security protocols, theorem proving, inductive methods.

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

Correspondence and offprint requests to: Yongjian Li, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, P.O. Box 8717, Beijing, China. E-mail: `lyj238@ios.ac.cn`.
The first author is supported by a grant 61170073 from National Natural Science Foundation of China.

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 also been the subject of many theoretical studies [39, 24, 23, 4, 21, 35, 37] and formal verification. The proposed definitions aim to capture either possibilistic [39, 24, 21, 35] or probabilistic [23, 4, 14] aspects of anonymity, and formal verification treats systems in different application domains, such as electronic voting systems [28, 13, 25], electronic cash protocols [32], file sharing [43, 11], and electronic healthcare [17, 18], etc. However, automatic approaches to the formal verification of anonymity have mostly focused on the model checking approach on systems with fixed configurations, e.g., [39, 40, 12, 14], while theorem proving is a more suitable approach when dealing with general systems of infinite state spaces [26]. We address this situation by investigating the possibility of using a powerful general-purpose theorem prover, Isabelle/HOL [33], to semi-automatically verify anonymity properties. In this paper, we focus on the framework of provable anonymity [21] and build a mechanical proof method for the analysis of anonymity protocols.

The starting point of the provable anonymity approach is the idea that various information hiding properties can best be expressed in the language of epistemic logic. This makes it possible to reason not only about the messages in a trace, but also about the knowledge agents gain from these messages. For instance, sender anonymity can typically be formulated as the receiver not knowing who sent a certain message. Central in the epistemic framework is the notion of observational equivalence of runs — a property holds for a protocol if it is true on every run which is indistinguishable for the intruder. 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 messages. In the framework proposed by Garcia et al. [21], observational equivalence is actually constructed via the *reinterpretation function*. Proving two traces equivalent essentially boils down to the existence of such a reinterpretation function. Within their framework, based on observational equivalence Garcia et al. define a few epistemic operators and use them to express information hiding properties such as *sender anonymity* and *unlinkability*

Although provable anonymity provides an elegant framework of anonymity in a uniform fashion, a deep gap exists between a pen-and-paper provable anonymity theory and a mechanized theory library. To the best of our knowledge, so far no one has formalized observation equivalence in the provable anonymity framework and the construction of this relation mechanically. Our work is essential not only to bridge this gap, but also to provide a mechanized proving methodology for anonymity protocols on the groundwork of provable anonymity theory.

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. In [21], the notion of observational equivalence is proposed without formalization in mind. Our proposal is a necessary step for its mechanical formalization.
- 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 [21].
  As mentioned before, in [21] no mechanical way is provided to construct a reinterpretation function. By our inductive approach to construct the alignment between the message sequences, we can construct such a reinterpretation function if it does exist. Furthermore, our work demonstrates that observational equivalence between traces cannot be obtained by simply lifting the message equivalence to the traces. Alignment between message sequences is required to make the constructed relation to be a bijection.

- We proceed to formalize anonymity properties in an epistemic logic framework as in [21]. Box and diamond operators are formalized at first, then sender anonymity and unlinkability are defined accordingly.
- We inductively define the semantics of anonymity protocols, e.g., Crowds and 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 via a method called *swap* messages. Formalizing the *swap* method and the corresponding proof techniques are novel in the sense that they give us a generally applicable way to construct an observationally equivalent trace.
- We build our theory in Isabelle/HOL [33] to have a mechanical framework for the analysis of anonymity protocols. We illustrate the feasibility of the mechanical framework through cases studies on Crowds [38] and Onion Routing [22, 41]. Based on our framework, we can inductively model an anonymity protocol, use epistemic operators to define anonymity properties, and follow the proof method to prove that the anonymity protocol under study satisfies the desired anonymity properties.

There are two guiding principles for us to choose Isabelle to formalize our anonymity theory. Firstly our induction techniques should be conveniently implemented by Isabelle/HOL. Isabelle/HOL is appropriate for this task because of its support for inductively defined sets and its automatic tool support. Secondly, our mechanized proofs correspond to their pen-and-paper counterparts as closely as possible. This means that for a user who has completed his proofs manually, little extra effort is required in order to let Isabelle/HOL check them.[1]

Note that this article is a revised and extended version of [30] that appears in the proceedings of the 6th International Conference on Availability, Reliability and Security. In this paper, we have developed a new notion of alignment between two message sequences. Combining the alignment requirement with the observational equivalence between two messages, we have proposed an improved definition of observational equivalence between two traces, which in turn leads to a revised formalization of provable anonymity in Isabelle/HOL. Thus, our framework can incorporate the concept of reinterpretation function which is extensively used in [21]. We have extended the case study on Onion Routing accordingly, and conducted a new case study on Crowds. As a consequence of the new notion of alignment, we also devote an additional subsection to prove the alignment requirement on observational equivalence between traces in the case study on Onion protocols. One of our previous work [31] also adopts an inductive approach to formalizing the strand space theory and implements the mechanical proof framework in Isabell/HOL. More specifically, the semantics of a protocol session is inductively formalized as a bundle in a strand space in [31], and classic trace properties such as secrecy and authentication are studied. In this work, the semantics of a protocol session is inductively formalized as a trace as well. Different from [31], our inductive approach is further developed to decide observational equivalence according to the adversary's knowledge obtained in two separate sessions, which is essential for defining anonymity properties.

Presentation of the paper. In this paper, we assume readers have some knowledge with Isabelle/HOL syntax. 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 summarizes related papers in the literature. Sect. 3 provides a preliminary introduction to notations and terminologies. Distinguishability and observational equivalence of messages are formally defined in Sect. 4. Then we introduce the notion of alignment for two sequences of messages in Sect. 5.

[1] Isabelle/HOL offers the Isar proof language [42], which is the abbreviation of the phrase "Intelligible semi-automated reasoning". Isar is an alternative proof language interface layer beyond traditional tactical scripts and readable for human beings.

Observational equivalence of traces is formally defined in Sect. 6. Epistemic operators and formalization of anonymity properties are presented in Sect. 7. We model and analyze Crowds and Onion Routing in Sect. 8 and Sect. 9, respectively. We conclude the paper in Sect. 10.

2. Related work

In this section, we summarize existing papers which are mostly related to our work. Besides the inherence and extension from the work on provable anonymity, our work is related to the work by Abadi and Cortier [2]. They introduce a notion of pattern equivalence which is also inductively defined to identify the observational equivalence between messages w.r.t. a set of keys. Furthermore, the equivalence corresponds to computationally indistinguishable ensembles. This definition is somehow restricted as the set of keys is only allowed to be symmetric, mainly because they focus on the computational soundness of the symbolic verification.

Another similar but more general, notion of message indistinguishability is static equivalence introduced in the applied pi calculus [3, 1]. Cryptographic primitives can be modeled using equational theory, thus more primitives can be incorporated in the framework, such as digital signature, XOR, etc. However this scheme is based on the applied pi calculus and cannot be easily formalized in a theorem prover.

More recently, the paper [10] studies process equivalence with length tests with respect to both passive and active intruders. A tool is developed for proving trace equivalence between two processes for a bounded number of sessions, based on the decision procedure of [9]. In addition, the tool ProVerif [5] can check observational equivalence defined in the applied pi calculus automatically [6, 8] with respect to active intruders. But it is not guaranteed to terminate and poses some restriction on the structure of processes.

In the literature, a proof method for checking branching bisimilarity [19, 20] is formalized in the theorem prover PVS. But this approach cannot be directly applied for security protocol analysis (see e.g., [34]) as it does not deal with message indistinguishability. Trace anonymity [39] is formalized using I/O automaton and the Larch prover is employed for check trace anonymity [26]. An anonymous fair exchange e-commerce protocol that is claimed to satisfy customer's anonymity is analyzed by the OTS/CafeOBJ method [27] following the approach proposed in [26]. However, these two works only consider a weaker intruder, which does not have the same ability to distinguish message sequences as we consider in this paper.

3. Preliminaries

3.1. 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:

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

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:

$$h ::= \mathsf{Agent}\ A \mid \mathsf{Nonce}\ N \mid \mathsf{Key}\ k \mid \mathsf{MPair}\ h_1\ h_2 \mid \mathsf{Crypt}\ k\ h$$

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

In an asymmetric key protocol model, an agent A has a public key $\mathsf{pubK}\ A$, which is known to all agents, and a private key $\mathsf{priK}\ A$. $\mathsf{pubK}\ A$ is the inverse key of $\mathsf{priK}\ A$, and vice versa. In a symmetric key model, each agent A has a long-term symmetric key $\mathsf{shrK}\ A$. The inverse key of $\mathsf{shrK}\ A$ 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 $\mathsf{shrK}\ A = \mathsf{shrK}\ A'$ then $A = A'$. In the following, we abbreviate $\mathsf{Crypt}\ k\ h$ as $\{\!|h|\!\}_k$, and $\mathsf{MPair}\ h_1 \ldots \mathsf{MPair}\ h_{n-1}\ h_n$ as $\{\!|h_1, \ldots, h_{n-1}, h_n|\!\}$. Such abbreviations are supported in Isabelle/HOL by syntax translation [33].

Operators parts, analz, and synth are inductively defined on a message set H. Their definitions are taken from [36] 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 the event ev, respectively. 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
```

The 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.

```
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
```

We define regularOrig $m\ tr$ to represent that a message m is 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.

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

3.2. Intruder model

We discuss anonymity properties based on observations of the intruder. In this section, we explain our intruder model. Dolev-Yao intruder model [16] 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 [36], 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)
```

4. 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 $m = \mathsf{Agent}\ A$ and $m' = \mathsf{Nonce}\ n$, 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 plain texts.

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 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, $Kn =$ synth (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 $m' = \{\!|\mathsf{Nonce}\ n'|\!\}_{\mathsf{pubK}\ B}$, with $n \neq n'$. We also assume $Kn =$ synth (analz{Key (priK B), m, m'}). We have $(m, m') \in$ Diff Kn by applying rule `basic` and rule `CryptDiff`.

In Example 1, the two messages m and m' are two encrypted messages by the same key pubK B, if the key pubK B is in Kn, then the two messages can be decrypted and distinguished.

Example 2. Let $n_0' \neq n$, $n_0' \neq n'$, $n \neq n'$, $A \neq B$, $n_0 \neq n$, $n_0 \neq n'$, $m =$ Nonce n, $m' =$ 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 $Kn =$ synth(analz $\{m_0, m_0', m_2, m_2'$, Key (pubK A), Key (pubK B), Key (priK B)$\}$), then we have $(m_2, m_2') \notin$ Diff Kn.

In Example 2, as priK B, m_0 and m_0' are in Kn, thus 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 $n_0' \notin Kn$.

Table 1. Two non-alignment message sequences

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

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, which are captured by the following three lemmas.

Lemma 1. $\mathsf{msgEq}\ Kn\ m\ m$

Lemma 2. $\mathsf{msgEq}\ Kn\ m\ m' \Longrightarrow \mathsf{msgEq}\ Kn\ m'\ m$

Lemma 3. $[\![\mathsf{msgEq}\ Kn\ m_1\ m_2; \mathsf{msgEq}\ Kn\ m_2\ m_3]\!] \Longrightarrow \mathsf{msgEq}\ Kn\ m_1\ m_3$

5. Alignment between Message Sequences

An immediate idea is to lift the observation equivalence directly to two message sequences by imposing the requirement that each corresponding message pair in the two sequences should be observationally equivalent. However, it is subtle to define the observational equivalence between two message sequences. For instance, there are two runs $msgSq$ and $msgSq'$ of a protocol, as shown in Table 1. Let $Kn = \mathsf{synth}(\mathsf{anlaz}\ \{m, m', m''\})$, $m \neq m'$. Even if we have $\mathsf{msgEq}\ Kn\ m\ m'$ and $\mathsf{msgEq}\ Kn\ m\ m''$, $msgSq$ 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 $msgSq'$.

In order to define observational equivalence between two traces (see Sect. 6), we propose an additional 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. Alignment requires that a message should have only one 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 separation) and synthesizing operation (e.g., encryption and concatenation) pair-wisely on the message pairs in the two message lists of the two message sequences. This matches well with the reinterpretation function as defined in [21].

From Examples 3 to 5, 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 3. If $\mathsf{priK}\ B$ and $\mathsf{priK}\ B'$ are not compromised, then $msgSq$ and $msgSq'$ as shown in Table 2 are different, 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 4. If $\mathsf{priK}\ B$ and $\mathsf{priK}\ B'$ and $\mathsf{priK}\ M$ are not compromised, then $msgSq$ and $msgSq'$ as shown in Table 3 are different, 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 pairwisely in the two message sequences, the intruder obtains a new pair of the following form $(\{\!|\{\!|\mathsf{Nonce}\ n_0'|\!\}_{\mathsf{pubK}\ B}|\!\}_{\mathsf{pubK}\ M}, \{\!|\{\!|\mathsf{Nonce}\ n_0'|\!\}_{\mathsf{pubK}\ B}|\!\}_{\mathsf{pubK}\ M})$. 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.)

Table 2. Two non-alignment message sequences for Example 3

$msgSq$	$msgSq'$
$\{\!\{\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}\}\!\}_{\mathsf{priK}\ M}$	$\{\!\{\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}\}\!\}_{\mathsf{priK}\ M}$
$\{\!\{\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}\}\!\}_{\mathsf{priK}\ M}$	$\{\!\{\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}\}\!\}_{\mathsf{priK}\ M}$
$\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}$	$\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}$
$\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}$	$\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}$

Table 3. Two non-alignment message sequences for Example 4

$msgSq$	$msgSq'$
$\{\!\{\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}\}\!\}_{\mathsf{pubK}\ M}$	$\{\!\{\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}\}\!\}_{\mathsf{pubK}\ M}$
$\{\!\{\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}\}\!\}_{\mathsf{pubK}\ M}$	$\{\!\{\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}\}\!\}_{\mathsf{pubK}\ M}$
$\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}$	$\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}$
$\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}$	$\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}$

Example 5. 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.

More formally, we first inductively define two more operators analz_pairs and synth_pairs to formalize the 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"
```

Table 4. Two alignment message sequences for Example 5

$msgSq$	$msgSq'$
$\{\!\{\mathsf{Nonce}\ n_0, \mathsf{Agent}\ B, \{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}\}\!\}_{\mathsf{pubK}\ M}$	$\{\!\{\mathsf{Nonce}\ n_1, \mathsf{Agent}\ B', \{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}\}\!\}_{\mathsf{pubK}\ M}$
$\{\!\{\mathsf{Nonce}\ n_1, \mathsf{Agent}\ B', \{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}\}\!\}_{\mathsf{pubK}\ M}$	$\{\!\{\mathsf{Nonce}\ n_0, \mathsf{Agent}\ B, \{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}\}\!\}_{\mathsf{pubK}\ M}$
$\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}$	$\{\!\{\mathsf{Nonce}\ n'_0\}\!\}_{\mathsf{pubK}\ B}$
$\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}$	$\{\!\{\mathsf{Nonce}\ n'_1\}\!\}_{\mathsf{pubK}\ B'}$

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 4. $[\![$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 $r\ Kn \longrightarrow ((m,m') \in r \longrightarrow m' = m'')\ ]\!] \Longrightarrow y = y'$

We cannot establish a similar result for the analz_pairs operator. For instance, let $r = \{(\{\!|$Nonce n, Nonce $n|\!\}$, $\{\!|$Nonce n, Nonce $n'|\!\})\}$. It is easy to verify that analz_pairs $r\ Kn$=$\{$(Nonce n, Nonce n), (Nonce n, 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.

6. 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 ($tr \approx_A tr'$), if the following conditions are satisfied:

- 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 pairs 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 1. In the work of Garcia et al. [21], a reinterpretation function between two message sequences is used as an 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 such a 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.

7. Epistemic Operators and Anonymity Properties

Using the observational equivalence relations over a trace set of possible worlds, we can formally introduce epistemic operators [21] 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.

7.1. 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 the following condition:

```
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.

7.2. 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.

All the definitions, lemmas, and proofs in this section are implemented in a formal theory `anonymity.thy`, which provides a mechanized library for anonymity protocol analysis. This theory comprises 2,462 lines. Its execution needs only 40 seconds. We make these Isabelle codes available at [29]. Our experiments are performed on the 64-bit Isabelle-2012 version, which is run on the Sugon 64-bit computing server platform which has a 160-multicore Intel Xeon CPU with 2.40GHz.

8. Case Study I: Crowds

The Crowds system [38] 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 [21]: 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 5. $[\![tr \in \mathsf{Crowds}; \mathsf{Says}\ R\ \mathsf{Server}\ (\mathsf{Nonce}\ n) \in \mathsf{set}\ tr]\!] \Longrightarrow$
$\exists A\ B.\mathsf{Says}\ A\ B\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\} \in \mathsf{set}\ tr$

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

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

Lemma 7. $[\![\ tr \in \mathsf{Crowds}; \mathsf{Says}\ R\ \mathsf{Server}\ (\mathsf{Nonce}\ n) \in \mathsf{set}\ tr]\!] \Longrightarrow \exists A.\mathsf{originates}\ A\ (\mathsf{Nonce}\ n)\ tr$

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

Suppose that the Server receives a request (identified by a nonce $\mathsf{Nonce}\ n$), 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 9. $[\![tr \in \mathsf{Crowds}; \mathsf{Says}\ R\ \mathsf{Server}\ (\mathsf{Nonce}\ n) \in \mathsf{set}\ tr]\!] \Longrightarrow$
$\mathsf{senderAnonymity}\ \{A.A \neq \mathsf{Server}\}\ \mathsf{Server}\ (\mathsf{Nonce}\ n)\ tr\ \mathsf{Crowds}$

Proof. By unfolding the definition of $\mathsf{senderAnonymity}$, for any agent X such that $X \neq \mathsf{Server}$, 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 7, 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 5, 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 6, 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 8, we have $tr' \in \mathsf{Crowds}$. By Lemma 7, $\mathsf{originates}\ X\ (\mathsf{Nonce}\ n)\ tr'$. Obviously, from the inductive definition of Crowds, we have $A' \neq \mathsf{Server}$. It is easy to verify that $\mathsf{view\ Server}\ tr' = \mathsf{view\ Server}\ tr$. Then we can derive $\mathsf{obsEquiv\ Server}\ tr\ tr'$. □

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 10.

Lemma 10. $[\![\ tr \in \mathsf{Crowds};\ tr = tr_1 @ (\mathsf{Says}\ A'\ B'\ \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\} \# tr_2);$
$(\forall A\ B.\mathsf{Says}\ A\ B \{\!|\mathsf{Agent\ Server}, \mathsf{Nonce}\ n|\!\} \notin \mathsf{set}\ tr_2)]\!] \Longrightarrow$
$\Box\ \mathsf{Spy}\ tr\ \mathsf{Crowds}\ (\mathsf{originates}\ A\ (\mathsf{Nonce}\ n))$

Analyzing the Crowds system only took us 5 days. The proof script comprises 695 lines, and executes in 20 seconds.

9. Case Study II: Onion Routing

Onion Routing [22, 41] provides both sender and receiver anonymity for communication over the Internet and servers as the basis of the Tor network [15]. Its main idea is based on Chaum's mix cascades [7] 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 router collects incoming messages until it has received k messages, and then permutes the messages and sends in batch to their intended receivers.

9.1. Modeling Onion Routing

In this paper, we model a simplified version of Onion Routing with only one onion router as done in [21]. 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 X M (Crypt (pubK M) ⦃Nonce n0,Agent Y,Crypt (pubK Y) (Nonce n)⦄) ∈ (set tr);
      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 the receiving status, X (or Y) and M are distinct, and both n_0 and n are fresh, an event $\mathsf{Says}\ X\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$ can be added. This step means that X sends a message to M which will later 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 later be simply discarded.
- If the length of the current trace is greater than or equal to i, this means that M is in the 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 our analysis, the intruder is passive in the sense that the spy (intruder) 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 for the message m, and replace them by dummy messages. Because all dummy messages will be discarded by the router, and only m will be peeled and forwarded to the intended receiver.

9.2. 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 property definitions, the proof is rather complicated. We give an overview of our formalization and the main 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 definitions of epistemic operators, which are used in the definition of sender anonymity and unlinkability, we need to construct another trace tr' which satisfies the following two conditions:

(1) tr' is still an Onion Routing trace, namely, $tr' \in \mathsf{oneOnionSession}\ i\ M$.
(2) tr' is observationally 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 conditions. The first two conditions 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 pairs obtained from tr and tr' after applying the operations analz_pairs and synth_pairs.

Sect. 9.4 formally introduces a function $\mathsf{swap}\ ma\ mb\ tr$, which servers the purpose of constructing such an equivalent trace tr'. Here ma, mb are the messages sent to the router in the trace tr. Sect. 9.4.1 gives its formal definition and proves simple correspondence properties of the swap function. Sect. 9.4.2 proves the first condition (1). Sect. 9.4.3 devotes to the proof of the condition (2-ii), and Sect. 9.4.4 proves the condition (2-i), then completes 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. 9.3. After these, we finish the proofs of the two anonymity properties in Sect. 9.5.

9.3. 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.

9.3.1. 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 exists an agent A who has sent $\{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$.

Lemma 11. $[\![tr \in \mathsf{oneOnionSession}\ i\ M;\ \mathsf{Says}\ M\ B\ \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y} \in \mathsf{set}\ tr]\!]$
$\Longrightarrow \exists n_0\ A.\mathsf{Says}\ A\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M} \in \mathsf{set}\ tr$

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

9.3.2. Uniqueness properties

Since an agent is required to originate fresh nonces when he sends a message to the router, thus 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 13. $[\![tr \in \mathsf{oneOnionSession}\ i\ M;\ \mathsf{Says}\ X\ M\ ma; \mathsf{Says}\ Y\ M\ mb]\!]$
$\Longrightarrow (X = Y \wedge ma = mb) \vee (\mathsf{noncesOf}\ ma \cap \mathsf{noncesOf}\ mb) = \emptyset$

From Lemma 13, 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 14. $[\![tr \in \mathsf{oneOnionSession}\ i\ M; \mathsf{Says}\ X\ M\ ma;\ X \neq Y;\ \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y} \sqsubset ma]\!]$
$\Longrightarrow \neg\mathsf{originates}\ Y\ (\{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y})\ tr$

As a consequence, 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 15. $[\![tr \in \mathsf{oneOnionSession}\ i\ M]\!] \Longrightarrow \mathsf{map\ msgPart}\ tr \in \mathsf{distinctList}$

With the above lemma, we can derive that the relation $(\mathsf{zip}\ (\mathsf{map\ msgPart}\ tr)\ sq)$ must be $\mathsf{single\_valued}$ if tr is in a trace of Onion Routing. Here, we use sq to indicate a message sequence of the same length.

Lemma 16. $[\![tr \in \mathsf{oneOnionSession}\ i\ M]\!] \Longrightarrow \mathsf{single\_valued}\ (\mathsf{zip}\ (\mathsf{map\ msgPart}\ tr)\ sq)$

9.3.3. Secrecy properties

First we need to introduce a new predicate:

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

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

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

Lemma 17. $[\![tr \in$ oneOnionSession i M; $M \notin$ bad; $B \notin$ bad; Says B M $ma \in tr$;
$ma = \{\!|$Nonce n_0, Agent Y, $\{\!|$Nonce $n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$; nonLeakMsg ma $M]\!]$
$\Longrightarrow$ Nonce $n_0 \notin$ analz (knows Spy tr)

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

Lemma 18. $[\![tr \in$ oneOnionSession i M; Says B $M\{\!|$Nonce $n_0|\!\}_{\mathsf{pubK}\ M} \in tr$; $M \notin$ bad; $B \notin$ bad$]\!]$
$\Longrightarrow$ Nonce $n_0 \notin$ analz (knows Spy tr)

9.4. 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.

9.4.1. The swap *function*

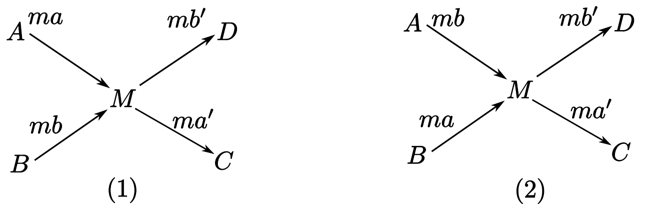
We first define a function swap ma mb tr, which returns another trace tr' satisfying the following conditions: (1) the sender and receiver of any event tr'_i in trace tr' are the same as in the corresponding event tr_i in tr; (2) the message of any event in tr' is swapped as mb if the message of the 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 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 formalized as the lemma below.

Lemma 19. Let tr be a trace.

1. $[\![(m_1, m_2) \in$ set (zip (map msgPart tr) (map msgPart (swap ma mb tr)))$]\!]$
   $\Longrightarrow m_1 = m_2 \vee (m_1, m_2) = (ma, mb) \vee (m_1, m_2) = (mb, ma)$
2. sendRecvMatchL tr (swap ma mb tr)
3. length (swap ma mb tr) = length tr
4. set (map msgpart tr) = set (map msgpart (swap mb ma tr))
5. swap ma mb tr = swap mb ma tr
6. $[\![$(Says X M $ma \in$ set tr)$]\!] \Longrightarrow$ Says X M $mb \in$ set (swap ma mb tr))
7. $[\![$(Says X M $mb \in$ set tr)$]\!] \Longrightarrow$ Says X M $ma \in$ set (swap ma mb tr))



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

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

Fig. 1. An illustration of function swap.

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. $[\![\mathsf{Says}\ A\ M\ ma \in tr; \mathsf{Says}\ B\ M\ mb \in tr; \forall ev. ev \in tr \longrightarrow (\exists\ A'\ B'\ m.\ ev = \mathsf{Says}\ A'\ B'\ m)]\!]$
$\Longrightarrow \mathsf{map\ msgPart}\ tr = \mathsf{map\ msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)$
11. $[\![\mathsf{Says}\ A\ M\ ma \in tr; \mathsf{Says}\ B\ M\ mb \in tr; \forall ev. ev \in tr \longrightarrow (\exists\ A'\ B'\ m.\ ev = \mathsf{Says}\ A'\ B'\ m)]\!]$
$\Longrightarrow \mathsf{knows\ Spy}\ tr = \mathsf{knows\ Spy}\ (\mathsf{swap}\ ma\ mb\ tr)$
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 $tr' =$ swap $ma\ mb\ tr$. In Lemma 19, 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 swap $ma\ mb\ tr$ has the same length as tr. Part 4 says that messages observed from tr is the same as those of swap $ma\ mb\ tr$. Part 5 shows that the trace swap $ma\ mb\ tr$ is the same as 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 Says $A\ M\ ma \in tr$, and Says $B\ M\ mb \in tr$, for Spy, the set of messages and knowledge obtained from tr is the same as those from swap $ma\ mb\ tr$. Part 12 says that nonces of mb will be disjoint from those of 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.

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

We first define a predicate nonceDisjUntil $ma\ tr$ stating that nonces of ma are disjoint with any other message occurring in any tr' such that tr' is a prefix of the 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 in tr, we have nonceDisjUntil $m\ tr$.

Lemma 20. $[\![tr \in \mathsf{oneOnionSession}\ i\ M; \mathsf{Says}\ A\ M\ m \in \mathsf{set}\ tr\ ]\!] \Longrightarrow \mathsf{nonceDisjUntil}\ m\ tr$

If nonceDisjUntil $m\ (ev\#tr)$, then nonceDisjUntil $m\ tr$. This is easily derived as the following lemma.

Lemma 21. $[\![\mathsf{nonceDisjUntil}\ m\ (ev\#tr)]\!] \Longrightarrow \mathsf{nonceDisjUntil}\ m\ tr$

The next predicate isRouterRecvMsg $m\ M$ specifies that m is a message sent to the router M. Within this subsection, when we mention ma and mb (see lemmas below), we always mean that they satisfy the predicates isRouterRecvMsg $ma\ M$ and isRouterRecvMsg $mb\ M$.

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

The 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 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 with a length $\geq i$, then swap ma mb tr is also a trace in oneOnionSession i M.

Lemma 22. $[\![tr \in \mathsf{oneOnionSession}\ i\ M;\ (\mathsf{noncesOf}\ ma) \cap (\mathsf{noncesOf}\ mb) = \emptyset;$
$\mathsf{isRouterRecvMsg}\ ma\ M;\ \mathsf{isRouterRecvMsg}\ mb\ M]\!] \Longrightarrow$
$\mathsf{bothContained}\ tr\ ma\ mb\ i\ M \longrightarrow \mathsf{nonceDisjUntil}\ ma\ tr\ i \longrightarrow \mathsf{nonceDisjUntil}\ mb\ tr\ i \longrightarrow \mathsf{swap}\ ma\ mb\ tr \in \mathsf{oneOnionSession}\ i\ M$

Proof. We apply induction to prove the lemma, and four subgoals will be generated. For notational convenience, we define the following abbreviations for the conclusion of this lemma:
$P \equiv \lambda tr.\mathsf{bothContained}\ tr\ ma\ mb\ i\ M \longrightarrow \mathsf{nonceDisjUntil}\ ma\ tr\ i \longrightarrow \mathsf{nonceDisjUntil}\ mb\ tr\ i \longrightarrow \mathsf{swap}\ ma\ mb\ tr \in \mathsf{oneOnionSession}\ i\ M$.

1. Base case onionNil: Obviously, swap ma mb $[] = []$, and $[]$ is in oneOnionSession i M.
2. Case onionCons1: Given a trace tr, agents X, Y, nonces n_0, n such that $tr \in \mathsf{oneOnionSession}\ i\ M$, $X \neq M$, $Y \neq M$, Nonce $n_0 \notin$ used tr, Nonce $n \notin$ used tr, and length $tr < i$, and the induction hypothesis (abbrv. *IH*) P tr, let $tr' = \mathsf{Says}\ X\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \mathsf{Crypt}\ (\mathsf{pubK}\ Y)\ (\mathsf{Nonce}\ n)|\!\}_{(\mathsf{pubK}\ M)}\#tr$, now we need to show P tr'. In order to prove this, we only need assume bothContained tr' ma mb i M, nonceDisjUntil ma tr' i and nonceDisjUntil mb tr' i, then show swap ma mb $tr' \in$ oneOnionSession i M. First, by length $tr < i$, bothContained tr ma mb i M trivially holds. From nonceDisjUntil ma tr' i, by Lemma 21 we also have nonceDisjUntil ma tr i. Similarly, we have nonceDisjUntil mb tr i. With P tr, we have (1) swap ma mb $tr \in$ oneOnionSession i M. Now we do case analysis on the newly added message $m' = \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \mathsf{Crypt}\ (\mathsf{pubK}\ Y)\ (\mathsf{Nonce}\ n)|\!\}_{(\mathsf{pubK}\ M)}$, there are three cases in total:
   (a) $m' = ma$. Notice that swap ma mb tr' = Says X M $mb\#$swap ma mb tr. We can easily show nonces $ma \cap$ used $tr = \emptyset$, because of nonces $ma = \{n, n_0\}$. With the premises (noncesOf ma) $\cap$ (noncesOf mb) $= \emptyset$, and nonceDisjUntil mb tr i, by Lemma 19-(12) we have (2) used (swap ma mb tr) $\cap$ nonces $mb = \emptyset$. From isRouterRecvMsg mb M, then we have two sub-cases:
      (a-1) either $mb = \{\!|\mathsf{Nonce}\ n_0', \mathsf{Agent}\ Y', \mathsf{Crypt}\ (\mathsf{pubK}\ Y')\ (\mathsf{Nonce}\ n')|\!\}_{(\mathsf{pubK}\ M)}$ for some n_0', n', Y' where $Y' \neq M$. From (2), we have Nonce $n_0' \notin$ used (swap ma mb tr), Nonce $n' \notin$ used (swap ma mb tr). With (1), by rule onionCons1, we can have swap ma mb $tr' \in$ oneOnionSession i M.
      (a-2) or $mb = \mathsf{Crypt}\ (\mathsf{pubK}\ M)(\mathsf{Nonce}\ n')$ for some n'. From (2), we have Nonce $n' \notin$ used (swap ma mb tr). With (1), by rule onionCons2, we can have swap ma mb $tr' \in$ oneOnionSession i M.
   (b) $m' = mb$. Similar to Case (a).
   (c) Neither $m' = ma$ nor $m' = mb$. From the premise nonceDisjUntil ma tr' i, we have (noncesOf ma) $\cap$ (noncesOf m') $= \emptyset$. Similarly, we have (noncesOf mb) $\cap$ (noncesOf m') $= \emptyset$. By Lemma 19-(13), we have nonces $m' \cap$ used (swap ma mb tr') $= \emptyset$. From this, we have Nonce $n_0 \notin$ used (swap ma mb tr), Nonce $n \notin$ used (swap ma mb tr). With (1), by rule onionCons1, we can have swap ma mb $tr' \in$ oneOnionSession i M.
3. Case onionCons2: Similar to Case onionCons1.

4. Case $\mathsf{onionCons3}$: Given a trace tr, agents X, Y, nonces n_0, n such that $tr \in \mathsf{oneOnionSession}\ i\ M$, $\mathsf{Says}\ X\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \mathsf{Crypt}\ (\mathsf{pubK}\ Y)\ (\mathsf{Nonce}\ n)|\!\}_{(\mathsf{pubK}\ M)} \in \mathsf{set}\ tr$, $\mathsf{Says}\ M\ Y\ \mathsf{Crypt}\ (\mathsf{pubK}\ Y)\ (\mathsf{Nonce}\ n) \notin \mathsf{set}\ tr$, $i \leq \mathsf{length}\ tr$, and induction hypothesis $P\ tr$. Now we need to show $P\ tr'$ where $tr' = \mathsf{Says}\ M\ Y\ \{\!|\mathsf{Nonce}\ n|\!\}_{(\mathsf{pubK}\ M)}\#tr$. In order to prove this, we only assume $\mathsf{bothContained}\ tr'\ ma\ mb\ i\ M$, $\mathsf{nonceDisjUntil}\ ma\ tr'\ i$ and $\mathsf{nonceDisjUntil}\ mb\ tr'\ i$, then show $\mathsf{swap}\ ma\ mb\ tr' \in \mathsf{oneOnionSession}\ i\ M$.
   Let $t' = \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \mathsf{Crypt}\ (\mathsf{pubK}\ Y)\ (\mathsf{Nonce}\ n)|\!\}_{(\mathsf{pubK}\ M)}$, and $t'' = \mathsf{Crypt}\ (\mathsf{pubK}\ Y)\ (\mathsf{Nonce}\ n)$. Notice that $\mathsf{isRouterRecvMsg}\ ma\ M$, and $\mathsf{isRouterRecvMsg}\ mb\ M$, then we have $t'' \neq ma$ and $t'' \neq mb$. Therefore, (3) $\mathsf{swap}\ ma\ mb\ tr' = \mathsf{Says}\ M\ Y\ t''\#\mathsf{swap}\ ma\ mb\ tr$.
   Furthermore from $t'' \neq ma$ and $t'' \neq mb$, and $\mathsf{bothContained}\ tr'\ ma\ mb\ i\ M$, we can easily have (4) $\mathsf{bothContained}\ tr\ ma\ mb\ i\ M$. From $\mathsf{nonceDisjUntil}\ ma\ tr'\ i$, by Lemma 21, we have $\mathsf{nonceDisjUntil}\ ma\ tr\ i$. Similarly, we have $\mathsf{nonceDisjUntil}\ mb\ tr\ i$. With $P\ tr$, we have (5) $\mathsf{swap}\ ma\ mb\ tr \in \mathsf{oneOnionSession}\ i\ M$. Now we perform case analysis on t', there are three cases in total:
   (a) $t' = ma$. From (4) and $i \leq \mathsf{length}\ tr$, we have $\mathsf{Says}\ X'\ M\ mb \in \mathsf{set}\ tr$ for some agent X', then by Lemma 19-(7), we have (6) $\mathsf{Says}\ X'\ M\ t' \in \mathsf{swap}\ ma\ mb\ tr$. From $\mathsf{Says}\ M\ Y\ t'' \notin \mathsf{set}\ tr$, by Lemma 19-(9), we have $\mathsf{Says}\ M\ Y\ t'' \notin \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr)$. With (5) (6), we can easily conclude $\mathsf{swap}\ ma\ mb\ tr' \in \mathsf{oneOnionSession}\ i\ M$.
   (b) $t' = mb$. Similar to Case (a).
   (c) Neither $t' = ma$ nor $t' = mb$. With $\mathsf{Says}\ X\ M\ t' \in \mathsf{set}\ tr$, by Lemma 19-(8), we have $\mathsf{Says}\ X\ M\ t' \in \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr)$. Similar to the argument in case (a), we have $\mathsf{Says}\ M\ Y\ t'' \notin \mathsf{set}\ (\mathsf{swap}\ ma\ mb\ tr)$. Therefore, with (5) (6), we can easily conclude $\mathsf{swap}\ ma\ mb\ tr' \in \mathsf{oneOnionSession}\ i\ M$.

□

Lemma 22 is rather complex, we must consider two cases: (1) $\mathsf{length}\ tr < i$; or (2) $\mathsf{length}\ tr \geq i$. For cases (1), we mainly need to consider when ma (mb) occurs in trace tr, then mb (ma) occurs in $\mathsf{swap}\ ma\ mb$. In order to make the $\mathsf{swap}\ ma\ mb$ satisfy the rule $\mathsf{onionCons1}$ or $\mathsf{onionCons2}$, mb (ma) should be in a form which is sent to the router M – these are guaranteed by conditions $\mathsf{isRouterRecvMsg}\ ma\ M$ and $\mathsf{isRouterRecvMsg}\ mb\ M$. Besides, the nonces created in mb (ma) should be fresh; this is the reason why premises $\mathsf{nonceDisjUntil}\ ma\ tr\ i$ and $\mathsf{nonceDisjUntil}\ mb\ tr\ i$ are needed. In fact, for case (1), we do not need to require that both ma and mb occur in tr, we can replace ma with a new message which contains fresh nonces which are disjoint with nonces of tr. However, for case (2), we have to require that both ma and mb should occur in tr, as ma occurs in tr and the peeled messages of ma and ma' also occur in tr. We notice that ma' also occurs in $\mathsf{swap}\ ma\ mb\ tr$ by the definition of swap, in order to make the $\mathsf{swap}\ ma\ mb$ satisfy the rule $\mathsf{onionCons3}$. We must require that ma should also occur in $\mathsf{swap}\ ma\ mb\ tr$, thus mb should also occur in tr.

Based on Lemma 20 and Lemma 22, we conclude an important result: for a trace $tr \in \mathsf{oneOnionSession}\ i\ M$, if both ma and mb are sent to the router M by some agent, then $\mathsf{swap}\ ma\ mb\ tr$ is still in $\mathsf{oneOnionSession}\ i\ M$. This lemma accurately captures the intuition of the function swap which is explained in Fig. 1.

Theorem 23. $[\![tr \in \mathsf{oneOnionSession}\ i\ M; \mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ ; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ ]\!]$
$\Longrightarrow \mathsf{swap}\ ma\ mb\ tr \in \mathsf{oneOnionSession}\ i\ M$

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 20, we have $\mathsf{nonceDisjUntil}\ ma\ tr\ i$ and $\mathsf{nonceDisjUntil}\ mb\ tr\ i$. By Lemma 22, we conclude that $\mathsf{swap}\ ma\ mb\ tr \in \mathsf{oneOnionSession}\ i\ M$. □

In fact, initially, we want to directly prove the lemma directly by induction. However, during the proof procedure, we find that we must face the case where only ma (or mb) occurs in tr, therefore we must strengthen the premises of the lemma, therefore $\mathsf{Says}\ A\ M\ ma(mb) \in \mathsf{set}\ tr$ is replaced by the premises $\mathsf{isRouterRecvMsg}\ ma(mb)\ M$ and $\mathsf{nonceDisjUntil}\ ma\ tr\ i$. When $\mathsf{length}\ tr \geq i$, we must require that both ma and mb must occur in tr because the swap function will not replace any message which is sent by the router. Therefore premises $\mathsf{bothContained}\ tr\ ma\ mb\ i\ M$ is added into premises. These are typical techniques involved when we use induction proof method. We need retune the induction assertion, and the key point is that the retuned one can be proved in each induction case.

9.4.3. Alignment properties

By Lemma 16, 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 24. $[\![tr \in (\mathsf{oneOnionSession}\ i\ M); r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)(\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)))]\!]$
$\Longrightarrow \mathsf{single\_valued}\ r$

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 24, we show that $\mathsf{analz\_pairs}\ r\ Kn$ is single_valued.

Lemma 25. $[\![r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)(\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)); Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows}\ \mathsf{Spy}\ tr)); r' = \mathsf{analz\_pairs}\ r\ Kn; M \notin \mathsf{bad}; tr \in \mathsf{oneOnionSession}\ i\ M;\ \mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr; (m, m') \in r'; (m, m'') \in r']\!]$
$\Longrightarrow m' = m''$

From Lemma 25, we derive a sufficient condition, as depicted in Lemma 4, in order to prove that $\mathsf{synth\_pairs}\ (\mathsf{analz\_pairs}\ r\ Kn)$ is single_valued.

Lemma 26. $[\![r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)(\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)); Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows}\ \mathsf{Spy}\ tr)); r' = \mathsf{analz\_pairs}\ r\ Kn;\ M \notin \mathsf{bad}; tr \in \mathsf{oneOnionSession}\ i\ M;\ \mathsf{nonLeakMsg}\ ma\ M; \mathsf{nonLeakMsg}\ mb\ M; \mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr; (m, m') \in r'; (m, m'') \in \mathsf{synth\_pairs}\ r'\ Kn]\!]$
$\Longrightarrow m' = m''$

Notice that $\mathsf{nonLeakMsg}\ ma\ M$ and $\mathsf{nonLeakMsg}\ mbM$ must be added in Lemma 26. Without the two conditions, ma (mb) can be synthesized from $\mathsf{Nonce}\ n$ if $ma = \{\!|\mathsf{Nonce}\ n, \mathsf{Agent}\ \mathsf{Spy}, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ \mathsf{Spy}}|\!\}_{\mathsf{pubK}\ M}$. Thus both (ma, ma) and (ma, mb) occur in $\mathsf{synth\_pairs}\ r'\ Kn$. By Lemma 25 and Lemma 4, we can conclude that $\mathsf{synth\_pairs}\ (\mathsf{analz\_pairs}\ r\ Kn)$ is single_valued.

Lemma 27. $[\![r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)(\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)); Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows}\ \mathsf{Spy}\ tr)); r' = \mathsf{analz\_pairs}\ r\ Kn;\ \mathsf{nonLeakMsg}\ ma\ M;\ \mathsf{nonLeakMsg}\ mb\ M;\ M \notin \mathsf{bad}; tr \in \mathsf{oneOnionSession}\ i\ M; \mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr;\ (m, m') \in \mathsf{synth\_pairs}\ r'\ Kn; (m, m'') \in \mathsf{synth\_pairs}\ r'\ Kn]\!]$
$\Longrightarrow m' = m''$

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

Lemma 28. $[\![r = \mathsf{set}\ (\mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)(\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)); Kn = \mathsf{synth}\ (\mathsf{analz}\ (\mathsf{knows}\ \mathsf{Spy}\ tr)); r' = \mathsf{analz\_pairs}\ r\ Kn;\ \mathsf{nonLeakMsg}\ ma\ M;\ \mathsf{nonLeakMsg}\ mb\ M;\ M \notin \mathsf{bad}; tr \in \mathsf{oneOnionSession}\ i\ M; \mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr;\ (m, m') \in (\mathsf{synth\_pairs}\ r'\ Kn)^{-1}; (m, m'') \in (\mathsf{synth\_pairs}\ r'\ Kn)^{-1}]\!]$
$\Longrightarrow m' = m''$

9.4.4. Observational equivalence between *tr* and swap *ma mb tr*

Let $r = \mathsf{zip}\ (\mathsf{map}\ \mathsf{msgPart}\ tr)\ (\mathsf{map}\ \mathsf{msgPart}\ (\mathsf{swap}\ ma\ mb\ tr)$ and $Kn = \mathsf{synth}\ (\mathsf{analz}\ ((\mathsf{knows}\ \mathsf{Spy}\ tr))$. For a pair $(ma, mb) \in\ r$, ma and mb are observationally equivalent w.r.t. Kn.

Lemma 29. $[\![tr \in \mathsf{oneOnionSession}\ i\ M; \mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr;\ \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr;\ A \notin \mathsf{bad};\ B \notin \mathsf{bad};$
$M \notin bad;\ ma = \mathsf{Crypt}\ (\mathsf{pubEK}\ M)\ \{\!|\mathsf{Nonce}\ na_0, \mathsf{Agent}\ Y, \mathsf{Crypt}\ (\mathsf{pubEK}\ Y)\ (\mathsf{Nonce}\ na)|\!\};$
$\mathsf{nonLeakMsg}\ ma\ M; \mathsf{nonLeakMsg}\ mb\ M]\!]$
$\Longrightarrow \mathsf{msgEq}\ (\mathsf{synth}\ (\mathsf{analz}\ ((\mathsf{knows}\ \mathsf{Spy}\ tr)))\ ma\ mb$

Notice that conditions $\mathsf{nonLeakMsg}\ ma\ M$ and $\mathsf{nonLeakMsg}\ mb\ M$ 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 $\mathsf{swap}\ ma\ mb\ tr$ is observationally equivalent to tr 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 $\mathsf{swap}\ ma\ mb\ tr$ in the view of the Spy.

Lemma 30. $[\![tr \in \mathsf{oneOnionSession}\ i\ M;\ ma = \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M};$
$\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr; \mathsf{Says}\ B\ M\ mb \in \mathsf{set}\ tr;\ A \notin \mathsf{bad}; M \notin \mathsf{bad}; B \notin \mathsf{bad};$
$\mathsf{nonLeakMsg}\ ma\ M;\ \mathsf{nonLeakMsg}\ mb\ M]\!]$
$\Longrightarrow \mathsf{obsEquiv}\ \mathsf{Spy}\ tr\ (\mathsf{swap}\ ma\ mb\ tr)$

Proof. By the definition of view, we can have (a) $\mathsf{view}\ \mathsf{Spy}\ tr = tr$ from $tr \in \mathsf{oneOnionSession}\ i\ M$. Unfolding the definition of obsEquiv, by part 3 in Lemma 19, we can prove (b) $\mathsf{length}\ (\mathsf{swap}\ ma\ mb\ tr) = \mathsf{length}\ tr$; by part 2 in Lemma 19, we also have (c) $\mathsf{sendRecvMatchL}\ tr\ (\mathsf{swap}\ ma\ mb\ tr$; by part 11 in Lemma 19, 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 1, we have $\mathsf{msgEq}\ Kn\ m_1\ m_2$; for case (2) and (3), they can be directly proved by Lemma 29. Let $r' = \mathsf{synth\_pairs}(\mathsf{analz\_pairs}\ r\ Kn)$, by Lemma 27 and 28, we have (f) $\mathsf{single\_valued}\ r'$ and $\mathsf{single\_valued}\ (r')^{-1}$.

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

9.5. Proving anonymity properties

Let us give two preliminary definitions: the senders in a trace is defined as $\mathsf{senders}\ tr\ M \equiv \{A.\exists m.\mathsf{Says}\ A\ M\ m \in \mathsf{set}\ tr\}$, 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 satisfying $\mathsf{nonLeakMsg}\ m\ tr$.

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

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 11, 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 12, 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 $\mathsf{Says}\ X\ M\ mb \in \mathsf{set}\ tr$, $X \neq M$, $X \notin \mathsf{bad}$. Let $ma = \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M}$. By $\mathsf{nonLeakTrace}\ tr\ M\ n$, we have both $\mathsf{nonLeakMsg}\ ma\ M$ and $\mathsf{nonLeakMsg}\ mb\ M$. Let $tr' = \mathsf{swap}\ ma\ mb\ tr$, by Lemma 30, we have $\mathsf{obsEquiv}\ Spy\ tr\ (\mathsf{swap}\ ma\ mb\ tr)$. By Lemma 23, we have $\mathsf{swap}\ ma\ mb\ tr \in \mathsf{oneOnionSession}\ i\ M$. From the fact $\mathsf{Says}\ X\ M\ mb \in \mathsf{set}\ tr$, by part 6 in Lemma 19, we have $\mathsf{Says}\ X\ M\ ma \in \mathsf{swap}\ ma\ mb\ tr$. By Lemma 12, we have $\mathsf{originates}\ X\ ma'\ (\mathsf{swap}\ ma\ mb\ tr)$. □

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

Proof. Let $runs = \mathsf{oneOnionSession}\ i\ M$, $AS\text{=}\ \mathsf{senders}\ tr\ M - \mathsf{bad}$. By unfolding the definition of the predicate unlinkability, we only need to prove that (1) $tr \models \Diamond \mathsf{Spy}\ runs\ (\neg \mathsf{originates}\ A\ ma'\ tr)$ and (2)$\mathsf{senderAnomity}\ AS$ $\mathsf{Spy}\ ma'\ tr\ runs$. Here (1) is our main goal, and (2) is proved in Lemma 31.

From the premise, there exist X and mx such that $\mathsf{Says}\ X\ M\ mx \in \mathsf{set}\ tr$, $X \neq A$, and $X \notin bad$. From

$\mathsf{Says}\ M\ B\ ma' \in \mathsf{set}\ tr$, by Lemma 11, there exists a message m, an agent A', a nonce n_0, such that ma has the form of $\mathsf{Says}\ A'\ M\ \{\!|\mathsf{Nonce}\ n_0, \mathsf{Agent}\ Y, \{\!|\mathsf{Nonce}\ n|\!\}_{\mathsf{pubK}\ Y}|\!\}_{\mathsf{pubK}\ M} \in \mathsf{set}\ tr$. Obviously, by the fact $\mathsf{regularOrig}\ ma'\ tr$, we have $A' \notin \mathsf{bad}$. In order to prove (1), by unfolding the definition of the diamond operator, we only need construct a trace tr' such that $\mathsf{obsEquiv}\ Spy\ tr\ tr'$ and $\neg\mathsf{originates}\ A\ ma'\ tr$. Here we do case analysis on A'.

If $A' \neq A$, then (1) can be proved immediately. Obviously $\mathsf{obsEquiv}\ Spy\ tr\ tr$, $tr \in \mathsf{oneOnionSession}\ i\ M$. By Lemma 14, we have $\neg\mathsf{originates}\ A\ ma'\ tr$. Otherwise, from $A' = A$, we have $X \neq A'$. let $tr' = \mathsf{swap}\ ma\ mx\ tr$, by Lemma 30, we have $\mathsf{obsEquiv}\ Spy\ tr\ tr'$. By Lemma 23, we have $tr' \in \mathsf{oneOnionSession}\ i\ M$. From $\mathsf{Says}\ X\ M\ mx \in \mathsf{set}\ tr$ and $\mathsf{Says}\ A\ M\ ma \in \mathsf{set}\ tr$, by Lemma 6, we have $\mathsf{Says}\ X\ M\ ma \in \mathsf{set}\ tr'$ and $\mathsf{Says}\ A\ M\ mx \in \mathsf{set}\ tr'$. From $X \neq A$, by Lemma 14, immediately we have $\neg\mathsf{originates}\ A\ ma'\ tr'$. □

Analyzing the Onion Routing protocol took us about one month's effort. This time is much longer because for this protocol we need more time to figure out a new proof technique. The proof script comprises 5,593 lines and executes in one minute.

9.6. A weakness of the protocol

Here, we show a weakness of the onion routing protocol, which is hinted by the premise $\mathsf{nonLeakTrace}\ tr\ M$. 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 analyze 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 $\{\!|\mathsf{Nonce}\ nb, \mathsf{Agent}\ \mathsf{Spy}, \{\!|\mathsf{Nonce}\ nb|\!\}_{\mathsf{pubK}\ \mathsf{Spy}}|\!\}_{\mathsf{pubK}\ M}$.

10. Conclusion

We have formalized the notion of provable anonymity in the theorem prover Isabelle/HOL. The inductive approach is one of the most important techniques we adopted. We proposed an inductive definition of message distinguishability based on the observer's knowledge, then defined message equivalence as the negation of message distinguishability. Next, we inductively define the alignment relation between message sequences, which is a key factor to construct a reinterpretation function on which the observational equivalence between traces is based. 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. The inductive approach helps us to define the semantics of observational equivalence, and protocol semantics as well. Correspondingly the inductive proof method is the most effective one to reason about the properties of the inductively defined semantics of anonymity protocols. Therefore, selecting a proper proof assistant is important which has built-in support for the inductive approach, which will make it feasible to mechanize all the theory in the theorem prover. Our choice is Isabelle, which plays a key role in our formalization.

When we prove that anonymity properties, e.g., sender anonymity, hold for a trace under consideration, we need manually to construct 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.

Acknowledgements. We would like to thank the anonymous referees who read carefully the first version of this paper and gave a lot of valuable comments. Those comments help us to greatly improve the quality of our paper both in content and presentation.

References

[1] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1-2):2–32, 2006.

[2] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 20(3):395, 2007.

[3] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th ACM Conference on Computer and Communications Security*, pages 16–25. ACM, 2005.

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

[5] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 82–96. IEEE CS, 2001.

[6] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

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

[8] V. Cheval and B. Blanchet. Proving more observational equivalences with ProVerif. In *Proc. 2nd POST*, volume 7796 of *LNCS*, pages 226–246. Springer, 2013.

[9] V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: negative tests and non-determinism. In *Proc. 18th ACM Conference on Computer and Communications Security*, pages 321–330. ACM, 2011.

[10] V. Cheval, V. Cortier, and A. Plet. Lengths may break privacy – or how to check for equivalences with length. In *Proc. 25th Conference on Computer Aided Verification*. Springer, 2013. To appear.

[11] 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. Springer, 2006.

[12] 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.

[13] 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.

[14] 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.

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

[16] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(12):198–208, 1983.

[17] 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.

[18] N. Dong, H. L. Jonker, and J. Pang. Enforcing privacy in the presence of others: Notions, formalisations and relations. In *Proc. 18th European Symposium on Research in Computer Security*, volume 8134 of *LNCS*, pages 499–516. Springer, 2013.

[19] W. J. Fokkink and J. Pang. Cones and foci for protocol verification revisited. In *Proc. 6th Conference on Foundations of Software Science and Computation Structures*, volume 2620 of *LNCS*, pages 267–281. Springer, 2003.

[20] W. J. Fokkink, J. Pang, and J. C. van de Pol. Cones and foci: A mechanical framework for protocol verification. *Formal Methods in System Design*, 29(1):1–31, 2006.

[21] 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.

[22] 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.

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

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

[25] 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.

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

[27] 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.

[28] 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.

[29] Y. Li. Formalization of provable anonymity in Isabelle. `http://lcs.ios.ac.cn/~lyj238/anonymity.html`.

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

[31] Y. Li and J. Pang. An inductive approach to strand spaces. *Formal Aspects of Computing*, 25(4):465–501, 2013.

[32] 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.

[33] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic.* LNCS 2283. Springer, 2002.
[34] J. Pang. Analysis of a security protocol in μCRL. In *Proc. 4th Conference on Formal Engineering Methods*, volume 2495 of *LNCS*, pages 396–400. Springer, 2002.
[35] 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.
[36] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.
[37] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management, April 2010.
[38] M. K. Reiter and A. D. Rubin. Crowds: anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.
[39] 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.
[40] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3/4):355–377, 2004.
[41] 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 CS, 1997.
[42] M. Wenzel. Isar - A generic interpretative approach to readable formal proof documents. In *Proc. 12th Conference on Theorem Proving in Higher Order Logics*, volume 1690 of *LNCS*, pages 167–184. Springer, 1999.
[43] 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 CS, 2004.

Appendix

We briefly present some Isabelle concepts, notations and commands, and our notation conventions. Readers might find these useful if they want to check our Isabelle proof scripts.

Isabelle's meta-logic is a fragment of Church's theory of simple types, which can be used to formalize an object-logic [33]. Normally, we use rich infrastructure of object-logics such as HOL to formalize some theory, which has been provided by the 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 strand space 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:

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

where $B_1, B_2, \ldots, B_n$ are the names of existing theories that T is based on. For our purpose, we only need to import HOL library `Main` to write the 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, $\mathsf{nat\ list}$ declares the type of lists whose elements 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. The type 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 also used in the object-logic HOL.

Introducing new types. There are three kinds of commands for introducing new types. $\mathsf{typedecl}$ *name* introduces new "opaque" type name without definition; types $name = \tau$ introduces an abbreviation name for type τ. $\mathsf{datatype}$ command can introduce a recursive data type. A general $\mathsf{datatype}$ definition is of the following 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 the 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. According to Isabelle's style, we use the notation lemma $name : [\![A_1; A_2; \ldots ; A_n]\!] \Longrightarrow B$ to denote that with assumptions $A_1, \ldots , A_n$ we can derive a conclusion B. Inline with Isar's style, a lemma is written as lemma $name$: assumes a_1 : "A_1" and ... and a_n : "A_n" shows B.

Proof scripts and proof states. In Isabelle's tactics-based style, a proof script comprises a sequence of applications of tactics: apply $tac_1, \ldots ,$ apply tac_n done. The script will be executed by Isabelle until all subgoals are solved. A typical proof in Isar has a human-readable structure as follows:

```
proof
assume $asm_0$ and ... and $asm_m$
have $formula_1$
   proof ....(*proof script for $formula_1$*) qed
...
have $formula_n$
   proof ....(*proof script for $formula_n$*) qed
have $formula_{n+1}$
   proof ....(*proof script for $formula_{n+1}$*) qed
qed
```

where $asm_0, \ldots , asm_m$ are assumptions, $formula_1, \ldots , formula_n$ are intermediate results. From the assumptions and intermediate results, we can show the final goal $formula_{n+1}$.

Techniques such as case distinction, induction, calculational reasoning in the Isar language can make our proof structured, which are immensely more readable and maintainable than apply-scripts. But the price we pay is that the length of proof in Isar is usually much longer than that of the counterpart in Isabelle tactical style. Therefore, we adopt a mixed style in our formalized proofs: we use commands in the Isar style to decompose a large goal into subgoals to keep our proof with a clear structure; when a subgoal is simple enough, we directly use apply-scripts to prove the subgoal, thus we can keep the length of our proof script relatively short. After processing a proof command, Isabelle will display a proof state:

$$\bigwedge x_1, \ldots , x_p.\ [\![A_1; A_2; \ldots ; A_n]\!] \Longrightarrow B$$

where $x_1, \ldots , x_p$ are local constants, $A_1, \ldots , A_n$ are local assumptions, and B is the actual (sub)goal in this proof state. Note that $\bigwedge$ is the universal quantifier in the meta-logic, instead of the conjunction operator ($\wedge$) in the object logic HOL.

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

tr, tr'	range over traces
m, ma, mb	range over messages
i, j	range over natural numbers for lengths of traces
A, B, X, M	range over agent names
k_1, k_2	range over keys
N, Na, Nb, n, na	range over nonces
Kn	range over message sets which represent knowledge of agents

A brief view of mechanical rule induction. Isabelle has built-in support for inductive approaches. After we type an inductive definition oneOnionSession as shown in Sect. 9.1, Isabelle will automatically generate an elimination rule oneOnionSession.induct as follows:

```
⟦ tr ∈ oneOnionSession i M;
   P [];
   ⋀ tr X Y n0 n.
   ⟦ tr ∈ oneOnionSession i M; P tr; X ≠ M; Y ≠ M; Nonce n0 ∉ used tr;
   Nonce n ∉ used tr; length tr < i ⟧
   ⟹ P (Says X M (Crypt (pubK M) ⦃Nonce n0, Agent Y, Crypt (pubK Y) (Nonce n)⦄)# tr);
   ⋀ tr X n.
   ⟦ tr ∈ oneOnionSession i M; P tr; X ≠ M; Nonce n ∉ used tr; length tr < i⟧
   ⟹ P (Says X M (Crypt (pubK M) (Nonce n)) # tr);
   ⋀ tr X n0 Y n.
   ⟦ tr ∈ oneOnionSession i M; P tr; i ≤ length tr;
   Says X M (Crypt (pubK M) ⦃Nonce n0, Agent Y, Crypt (pubK Y) (Nonce n)⦄) ∈ set tr;
   Says M Y (Crypt (pubK Y) (Nonce n)) ∉ set tr⟧
   ⟹ P (Says M Y (Crypt (pubK Y) (Nonce n)) # tr)⟧
⟹ P tr
```

oneOnionSession.induct formalizes an induction proof method on traces of an Onion Routing protocol. Namely, if we want to prove $P\ tr$ for any trace $tr \in$ oneOnionSession $M\ i$. We first need to prove P []; second we need to prove $P\ tr \Longrightarrow P\ (ev\#tr)$, where ev is an event ev added into the trace tr which is defined according to the rules onionCons1-onionCons3.

In the following discussions, we show how to formally prove Lemma 22 in Isabelle. First we need to write a lemma to specify the proof goal by a lemma command. For Lemma 22, we define it in Isabelle as follows:

```
lemma trace_invariant:
assumes a1:"tr:oneOnionSession k M" and a2:"(noncesOf ma) ∧ (noncesOf mb) =∅" and
a3:"isRouterRecvMsg ma M" and a4: "isRouterRecvMsg mb M"
shows " bothContained tr ma mb k M⟶ nonceDisjUntil ma tr k ⟶nonceDisjUntil mb tr k
⟶ (swap ma mb tr: oneOnionSession k M)" (is "P tr")
```

After defining the lemma, we need to ask Isabelle to assist us to prove the goal, and use the command using a1 proof induct to apply the induction rule oneOnionSession.induct to solve the goal. The premise $tr \in$ oneOnionSession $k\ M$ will be eliminated. Note that the property P under consideration in this lemma is "$\lambda tr.$bothContained $tr\ ma\ mb\ k\ M \longrightarrow$ nonceDisjUntil $ma\ trk \longrightarrow$ nonceDisjUntil $mb\ tr\ k \longrightarrow$ swap $ma\ mb\ tr \in$ oneOnionSession $k\ M$". According to the definition of oneOnionSession.induct, four proof goals will be automatically generated by Isabelle. Here we only present the second one:

```
2. ⋀tr X Y n0 n.
⟦tr ∈ oneOnionSession k M; noncesOf ma ∧ noncesOf mb = ∅;
isRouterRecvMsg ma M; isRouterRecvMsg mb M;
bothContained tr ma mb k M⟶nonceDisjUntil ma tr k ⟶ nonceDisjUntil mb tr k
⟶ swap ma mb tr ∈ oneOnionSession k M;
X≠M; Y≠M; Nonce n0 ∉ used tr; Nonce n ∉ used tr; length tr < k⟧ ⟹
bothContained(Says X M (⦃Nonce n0, Agent Y, Crypt (pubK Y) (Nonce n)⦄_(pubK M)#tr)ma mb k M
⟶nonceDisjUntil ma(Says X M ⦃Nonce n0, Agent Y, Crypt (pubK Y) (Nonce n)⦄_(pubK M)#tr) k
⟶nonceDisjUntil mb(Says X M ⦃Nonce n0, Agent Y, Crypt (pubK Y) (Nonce n)⦄_(pubK M)#tr) k
⟶swap ma mb(Says X M ⦃Nonce n0, Agent Y, Crypt (pubK Y) (Nonce n)⦄_(pubK M)#tr)
∈ oneOnionSession k M
```

Notice that the fifth premise is the induction hypothesis $P\ tr$, and the conclusion to be shown is P (Says $X\ M$ ⦃Nonce $n0$, Agent Y, Crypt (pubK Y) (Nonce n)⦄$_{(\mathsf{pubK}\ M)}\#tr$). From the above, we can see that this lemma is not trivial, and the proof obligations involved are rather complex. Isabelle can help us in an induction proof: selecting proper induction rule to execute, generating subgoals for base case and induction steps. It is error-prone for human to perform these tasks. However, Isabelle/Isar can automatically finish these tasks in a mechanical way. After the subgoals are created, our proof structure is naturally decomposed into four parts: one for the base case onionNil, the other three for the induction cases: onionCons1-onionCons3.

Besides, for the typical proof techniques of using case analysis as shown in the proof of Lemma 22, we use Isar commands for calculational reasoning to perform the proof of all case analysis. Thanks to the Isar proof script language provided by Isabelle, the mechanical proof structure has a similar structure as the one shown in Sect. 9.4.2.