X-Men: A Mutation-Based Approach for the Formal Analysis of Security Ceremonies

Diego Sempreboni
Department of Informatics
King's College London
London, UK
diego.sempreboni@kcl.ac.uk

Luca Viganò
Department of Informatics
King's College London
London, UK
luca.vigano@kcl.ac.uk

***Abstract*—There is an increasing number of cyber-systems (e.g., payment, transportation, voting, critical-infrastructure systems) whose security depends intrinsically on human users. A security ceremony expands a security protocol with everything that is considered out-of-band to it, including, in particular, the mistakes that human users might make when participating actively in the security ceremony. In this paper, we introduce a novel approach for the formal analysis of security ceremonies. Our approach defines mutation rules that model possible behaviors of a human user, and automatically generates mutations in the behavior of the other agents of the ceremony to match the human-induced mutations. This allows for the analysis of the original ceremony specification and its possible mutations, which may include the way in which the ceremony has actually been implemented. To automate our approach, we have developed the tool X-Men, which is a prototype that extends Tamarin, one of the most common tools for the automatic unbounded verification of security protocols. As a proof of concept, we have applied our approach to two real-life case studies, uncovering a number of concrete vulnerabilities.**

***Index Terms*—Security ceremonies, socio-technical security, formal methods, mutations**

1. Introduction

Context and Motivation. Ellison [18] introduced the concept of *security ceremony* as an extension of the concept of security protocol, with human nodes alongside computer nodes and with communication links that include UI, human-to-human communication and transfers of physical objects that carry data. In particular, Ellison remarked that "what is out-of-band to a protocol is in-band to a ceremony, and therefore subject to design and analysis using variants of the same mature techniques used for the design and analysis of protocols".

However, in contrast to security protocol analysis, for which a plethora of mature approaches and tools exist, *security ceremony analysis* is a discipline that is still in its childhood, with no widely recognized methodologies or comprehensive toolsets. State-of-the-art approaches and tools for security protocol analysis (e.g., [3], [10], [19], [26], [28]) cannot be directly employed for security ceremonies as they take a "black&white" view and formalize protocols by

- considering one or more attackers that can carry out whatever actions they are able to in order to attack the protocol, but then
- modeling all other protocols actors (regardless of whether they are computers or human users) as honest processes that behave according to the protocol specification.

When considering security ceremonies, in which humans are first-class actors, it is not enough to take this "black&white" view. It is not enough to model human users as "honest processes" or as attackers, because they are neither. Modeling a person's behavior is not simple and requires formalizing the human "shades of gray" that such approaches are not able to express nor reason about. It requires modeling the way humans interact with the protocols, their behavior and the mistakes they may make, independent of attacks and, in fact, independent of the presence of an attacker.

Some preliminary approaches have been proposed for security ceremony analysis (e.g., [6], [7], [9], [12], [14], [23], [24], [31], [32]), but they have barely skimmed the surface of taking into account human behavioral and cognitive aspects in their relation with "machine" security.

Contributions. In this paper, we introduce a novel approach for the formal analysis of security ceremonies that focuses on the vulnerabilities that result from the mistakes that human users might make. More specifically, we provide three main contributions.

1. Formalization. We define a formal approach that allows security analysts to model possible mistakes by human users as *mutations* with respect to the behavior that the ceremony originally specified for such users. We focus on three main human mutations of a ceremony,

- *skipping one or more of the actions that the ceremony expects the human user to carry out* (such as sending or receiving a message),
- *replacing a message with another one*,
- *adding an action*,

and their combinations (but our approach is open to extensions with other mutations).

Human ceremony mutations will likely have an effect also on the other agents of the ceremony, honest or malicious as they may be. There are two cases: (1) the other agents are able to reply to a human mutation because

© 2020, Diego Sempreboni. Under license to IEEE.
DOI 10.1109/EuroSP48549.2020.00014

the changes are not too relevant or because the ceremony has somehow made provision for it (e.g., by an if-the-else that captures both original and mutated human behavior), or (2) the other agents are not able to reply to a human mutation. To investigate whether this human mutation may lead to an attack, we formalize *algorithms for human mutations* and *algorithms for matching mutations* for the other agents, which allow us to create a complete mutated ceremony specification that can be executed and analyzed for vulnerabilities. Our algorithms allow for the analysis of the original ceremony specification and its possible mutations, which may include the way in which the ceremony has actually been implemented.[1]

2. Tool. We have developed a prototype tool called *X-Men* (the name was chosen to suggest that we consider human mutations), which, as shown in Figure 1, creates mutated models that can then be input to Tamarin [26], one of the most advanced tools for the automatic unbounded verification of security protocols. X-Men can be used with human mutations only (without matching) but this will often yield non-executable specifications as the non-human roles simply won't reply (thus thwarting attacks caused by the human mutation). Matching mutations adjust non-human roles so that they can be executed together with a mutated human role. They are implemented automatically by X-Men, which generates the matching mutation from the protocol specification based on the human mutation (it changes the non-human-role specification to receive/send messages according to the human mutation) and propagates mutations to create an executable trace that can be analyzed in search for attacks. These attacks might be real attacks on the ceremony's (specification and) implementation, or be just the result of the mutations and not be applicable on the actual implementation. In the spirit of *mutation testing* [11], [15], [17], [22], the attacks discovered by X-Men could be used to generate and apply test cases for the ceremony implementation, but we leave this extension of X-Men for future work.

3. Proof-of-concept. We have applied our approach to two real-life case studies, the Oyster ceremony and the SAML-based Single Sign-on for Google Apps [4], uncovering a number of concrete vulnerabilities, which had so far been discovered only by empirical observation of the actual ceremony execution or by directly formalizing alternative specifications of the ceremony by hand. Instead, X-Men allowed us to generate them automatically.

Organization. In § 2, we introduce our motivating and running example. In § 3, we describe the intuitions that underlie our approach. In § 4, we describe how we formally model security ceremonies and their mutations. In § 5, we describe X-Men and its proof-of-concept. In § 6, we discuss related work. In § 7, we draw conclusions and discuss future work. Additional information is provided in the appendices. All our formal models and the code of X-Men are available at [40].

1. It is often the case that the implementation of a protocol or ceremony deviates from the original specification. There are several possible reasons for this. For instance, the implementation might have deviated from the specification in order to accommodate initially unforeseen behavior by the human users (and this might actually be one of the reasons for the issues in our first case study, the Oyster ceremony) or simply because the implementers did some mistakes (as in our second case study, the SAML-based Single Sign-on for Google Apps [4]).

2. An Example: The Oyster Card Ceremony

We will use the Oyster Card ceremony as a motivating and running example. The *Oyster Card* (or just Oyster, for short) is a plastic credit-card-sized, rechargeable, stored-value, contactless smartcard used on public transport in Greater London in the United Kingdom. The Oyster can hold pay-as-you-go credit, travelcards and passes for underground and overground trains, buses and trams. It is promoted by *Transport for London (TfL)* and since its introduction in June 2003, more than 86 million cards have been used [38]. Similar systems are in use in a large number of other countries in almost all continents, and, interestingly, most of them suffer from problems similar to the ones of the Oyster that we will discuss below.

As shown in Figure 2a, the Oyster is used by touching it on an electronic reader when entering and leaving the transport system in order to validate it or deduct funds. Actually, this touch-in/touch-out is part of the ceremony used on the London underground (nicknamed the *Tube*) and trains, which is what we focus on in this paper, whereas on London buses passengers touch in their Oyster only when boarding (instead, in Sydney, Australia, passengers are required also to touch out when they alight the bus). Figure 2b shows an entrance/exit gate of the Tube.

Figure 3a gives a Message Sequence Chart (MSC) of the main Oyster Ceremony for the Tube, which is carried out by 3 roles: the human passenger H, the entrance gate $GateIn$ and the exit gate $GateOut$.

1) The human passenger H touches their Oyster on the reader at the entrance gate, which amounts to H sending the Oyster number $oyster$ to $GateIn$.
2) The reader writes an identifier on the Oyster, which amounts to $GateIn$ replying with the message $oyster, gin$, where gin is the identifier of $GateIn$.
3) At the end of the journey, the passenger touches the Oyster on the reader at the exit gate, which amounts to H sending to $GateOut$ the number $oyster$, the current $balance$ of the card and gin.
4) $GateOut$ calculates the journey fare based on the distance traveled from $GateIn$, subtracts the amount from the card's balance, and sends to H the new balance along with the card number and a $finish$ flag.

Some remarks are in order. First of all, note that we did not obtain this specification from TfL, with whom we have not been in touch, but rather we modeled our own experience of using the Oyster. This is fine as we do not need our example to be real but rather realistic enough to showcase the main features of our approach; still, the vulnerabilities that we identify are actual problems that the real Oyster system suffers from.

Second, even though the Oyster is based on the MiFare chip, which in its first version (Mifare Classic family) used the proprietary encryption algorithm Crypto-1, our specification does not use any kind of encryption for the messages. This does not represent a lack of accuracy as we actually aim to model the ceremony in a way that is

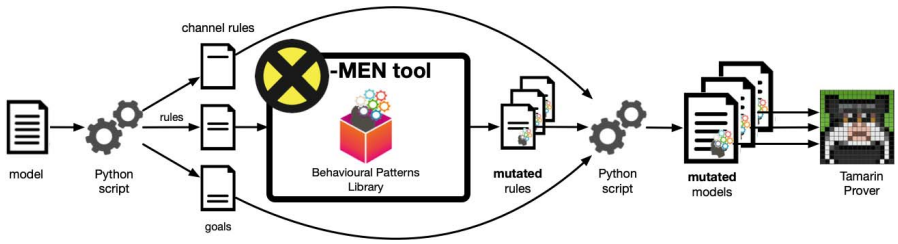


Figure 1: The workflow of the X-Men tool: from models to mutated models that are input to Tamarin

(a) Touching the Oyster on an Electronic Card Reader

(b) A Gate of the Tube

Figure 2: Using the Oyster Card in the Tube

independent of the low-level cryptographic details, thereby also keeping in mind that our approach focuses on what is under direct influence and control of the human, and cryptography most likely is not. However, it would not be difficult to include encryption and decryption in our specifications, and in fact the language that we describe below does contain cryptographic operators.[2]

Third, we focused only on the core message-passing of the ceremony and did not include the information that is displayed on the screens that are placed above the gate's reader, which show, e.g., the credit on the card when entering and exiting and the fare of the trip when exiting.

Fourth, the ceremony in Figure 3a is actually one of the possible ceremonies that could be considered for the use of the Oyster and several variants could be modeled, such as: a ceremony in which the reader at the exit gate does not immediately synchronize with the system, a ceremony in which the passenger does not have enough credit for the entrance gate to open (if the Oyster's balance is too low, the gate would display a message to the passenger asking them to top up the credit on the card), or a ceremony in which the passenger changes from an overground train to an underground train or vice versa, and thereby touches the Oyster at an intermediate gate to register the change of train. Again, we aim to be realistic rather than real and, in fact, our approach generalizes to these variants quite straightforwardly.

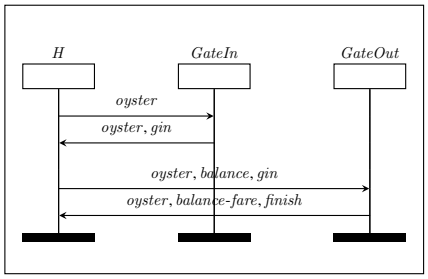
Finally, passengers are nowadays able to pay not only with the Oyster but also with a contactless credit or debit card (possibly associated with an Apple Pay or Google Pay device). In that case, the ceremony is the same as the one in Figure 3a but without the $balance$ and replacing $oyster$ with the number of the contactless credit/debit card (the physical one used to touch in/out or the one associated with Apple or Google Pay). To avoid having to distinguish the two cases, let us introduce a generalized ceremony for the Tube, which passengers can carry out with either their Oyster or a contactless card, as shown in Figure 3b. Here, we use a public unary function bal that computes the current balance of an Oyster or simply sends a message "$accept$" in case of a contactless card. This is what H sends in the third message, and then $GateOut$ replies in the final message by sending $bal(card)'$, which is the updated balance of the Oyster or another "$accept$" message, respectively.

Before we continue with the discussion of how we formally model security ceremonies in our approach, let us return to Figure 2a, where the sticker beside the reader reminds passengers to always touch in and out. In fact, the London underground is quite full of posters like the ones in Figure 4. The poster on the left of Figure 4 reminds passengers that in order to pay the right fare, they need to touch in at the start and touch out at the end of all journeys; if they do not, then TfL will not know where the passenger has traveled, so they cannot charge the right fare for the journey. This is called an *incomplete journey* and the passenger could be charged a maximum fare ranging between £8.00 and £19.80 [37]. Passengers who do not touch in at the start of a journey are also liable to pay a penalty fare (or could even be prosecuted).

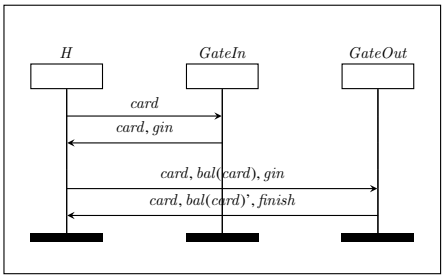
The poster on the right of Figure 4 warns passengers that if they touch on a reader their purse or wallet containing two or more cards (be they Oyster cards or contactless payment cards), then they could experience *card clash* [36]. This means that when the card reader detects two cards, it could take payment from a card that the passenger did not intend to pay with, or, more dangerously, that the passenger could be charged two fares for his journey or even two maximum fares for his journey (this happens when a passenger mistakenly touches in with one card and touches out with another card, resulting in two incomplete journeys).

It is interesting to observe that, in both these cases, security is "pushed" from the system to the human user. But humans do mistakes and this might endanger their security, which here means that they possibly have to pay considerably more than they should. Our approach allows us to show (in a formal and automated way) that indeed if passengers forget to touch in or out, or touch with two or more cards at the same time, then they will be billed unfairly. Let us thus proceed by explaining how we formally model and reason about security ceremonies.

2. Note also that initial versions of the MiFare chip, and thus of the Oyster, suffered from a number of attacks [13], [16], [21], but the current version of the Oyster does not suffer from these problems anymore since it is based on the new MiFare DESFire family that uses stronger encryption algorithms.



(a) The Main Oyster Ceremony for the Tube



(b) The Generalized Main Ceremony for the Tube

Figure 3: The Ceremonies for the Tube

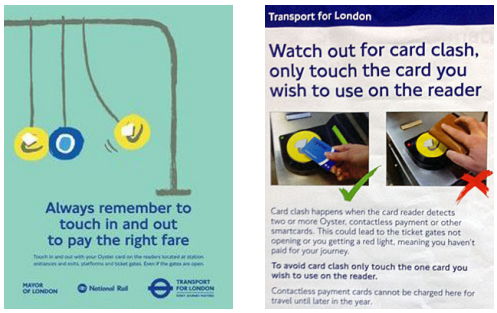


Figure 4: Warnings issued to the Tube passengers

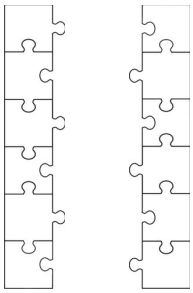


Figure 5: A simple ceremony between a User (left) and a System (right) depicted as a jigsaw puzzle

3. Our approach in a nutshell

The standard way to formally model and analyze a security protocol/ceremony is to formalize how agents (attempt to) execute the *roles* of the protocol/ceremony to achieve one or more security goals in the presence of an attacker. Roles are sequences of events (sending or receiving messages, generating fresh values, etc.), which are usually represented graphically by a structure generated by causal interaction such as *strands* [20] or the vertical lines in MSCs and *Alice&Bob notation* [2], or less graphically by a process in a process algebra such as in the *applied pi calculus* [1]. In Tamarin (and thus in the X-Men tool), a role is formalized by a so-called *role script*, which is basically the projection to an individual role of an extended Alice&Bob specification, and corresponds to a strand or an applied pi calculus process.

We can represent this graphically by viewing the roles/strands of a ceremony as separate lines of assembled jigsaw puzzle pieces that can be connected with each other as shown in the example in Figure 5. When complete, the jigsaw puzzle produces a complete picture: the run of the ceremony.

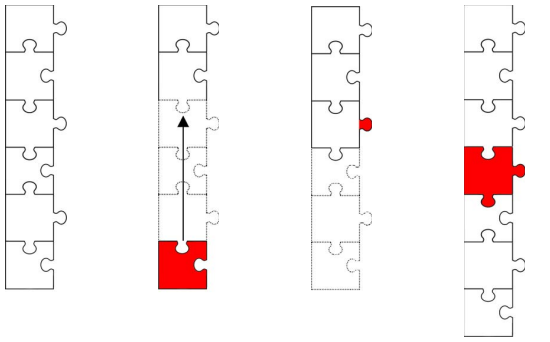
Now, we have all been there: you are trying to assemble one of those really difficult jigsaw puzzles, you know, one of those where the resulting image is so complex that it is difficult to understand which pieces you should actually interlock. You start from the borders, trying to complete at least one line and proceed from there, but even that is proving to be difficult as you do not understand which pieces do really fit together. So, what do you do? You try. You try to interlock pieces that appear to fit together even though this will turn out to be wrong as they will not allow you to produce the desired image — but you do not know that yet. Or maybe you simply do a mistake and append a piece that does not belong there.

This is illustrated by Figure 6: the human user could append a wrong piece pictured in red as in Figure 6c, which raises the question of how the two remaining pieces would fit (they are thus drawn with dotted lines), or the human could not know how long the edge should be and terminate it by attaching the piece pictured in red as in Figure 6b; or the human could add one more piece to the edge as in Figure 6d.

Returning to our running example, the human user might not fully understand the ceremony role that he is supposed to carry out and

- *skip* some intermediate actions, e.g., touching out with an Oyster without having touched in with any card, as illustrated in Figure 6b by the anticipated termination of the role;
- *replace* an action with another, e.g., using a contactless credit card to touch out instead of the Oyster he used to touch in, as illustrated in Figure 6c by the different outgoing connector, which represents a different message being sent;
- *add* some actions, e.g., touch in with two cards, as illustrated in Figure 6d by the additional piece.

In our approach, we represent these human "mistakes" as *mutations* with respect to the role as specified originally — hence the name "X-Men" for our tool, which captures the fact that we are considering mutations of the original human behavior. Such a mutation does not just have a local effect (for that event of the role) but will likely have an effect on the subsequent events in the role, which we



(a) The role User as it was specified

(b) The role User as carried out by a human who connects a piece to terminate the role sooner than specified

(c) The role User as carried out by a human who connects a different piece than the one specified

(d) The role User as carried out by a human who connects a piece to extend the length of the role

Figure 6: A human carrying out the role User... and mutating it, by mistake or lack of understanding

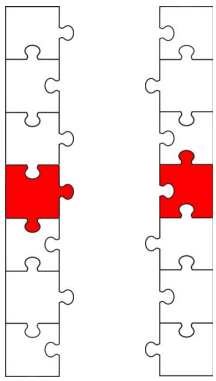


Figure 7: The "add" mutation of the role User (as in Figure 6d) and the matching mutation of the role System

illustrated by drawing the subsequent puzzle pieces with dotted lines. This is because the knowledge of the human agent will likely change depending on what has really happened.

It is, however, not enough to simply allow the human to carry out these unforeseen actions (add, skip or replace some parts of the role). In order to reason about what would happen if the human carried out these mutations, we need to capture the fact that a mutation of the human behavior will likely have an effect also on the other agents of the ceremony. More specifically, consider again, for simplicity, the ceremony between User and System in Figure 5 and consider the scenario in which a human playing the role User replaces an event of his role with a different one, i.e., sends a message m' instead of the specified message m, as depicted in Figure 6c, which is a mutation of Figure 6a. There are two cases.

In the first case, the System is able to reply to m'. This means that the System can still receive (and "understand") and reply to m' because the changes with respect to m are not too relevant. For instance, this might happen when the ceremony does not provide the System with enough information to check the content of m', e.g., when the User sends a contactless card number instead of an Oyster card number but the System does not have previous information that allows it to check whether it received the correct card number, or when the message has been encrypted with a symmetric key that the System does not (yet) possess. In this case, we can carry on with our analysis of the ceremony to check whether either the original or the mutated User role lead to an attack.

In the second case, the System is not able to reply to m' as that mutation is not envisioned by the System's role as specified by the original ceremony. But what about the ceremony's implementation? Does the implementation really conform to the specification? If it does, then the implementation of the System role will not reply and we are fine as the run with the mutation m' will not terminate. But what if the ceremony's developers, after they designed the specification and/or deployed the implementation, realized that the User could indeed send a different message (or skip some actions or add some) and made provisions for this case? For instance, they could have introduced in the implementation an "if-then-else" that captures both m and m', i.e.: "if you receive m then reply with message n else if you receive an $m' \neq m$ then reply with message n'".[3] To reason about such a situation, we can use the mutation as a test case that is relevant for the ceremony's implementation. We pair the mutation of the User role with a *matching mutation* of the System role to generate an executable trace of the ceremony. This is in line with *mutation testing* [11], [15], [17], [22], which is an approach to design software tests where mutants are based on well-defined mutation operators that either mimic typical programming errors (such as using the wrong operator or variable name) or force the creation of valuable tests (such as dividing each expression by zero). In our approach, mutants are based on mutation operators that mimic typical human mistakes (add, skip or replace, as discussed above) and force the creation of mutations in the other ceremony agents to match the human mutation. For concreteness, for the ceremony between User and System in Figure 5, our approach mutates the role of the System as shown in Figure 7 to match the human User's replace mutation of Figure 6c. The mutation of the step of the System to match the mutated step of the human User possibly entails a mutation of the subsequent steps of the System role, which we again illustrate with dotted lines.

3. Note that this does not mean that the User is fully aware of this. The User might just be aware of (or have been instructed about) the "then branch" of the System's role, which captures the User's normal behavior; think of the Oyster User who follows the touch-in-touch-out ceremony as expected. Hence, the User might, unknowingly and unwillingly, fall into the "else branch" of the System's role (e.g., by touching out with a contactless card instead of the Oyster card that was used for touch in) and thus be billed much more than expected. The problem with these "else branches" is that they often were not present in the original specification of the whole ceremony and were added to the implementation as an afterthought, after having observed the "wrong" behavior of users, as was likely the case for the Oyster ceremony. Warnings like the ones in Figure 4 are meant to alert the users about the "else branch" of the ceremony. We believe that rather than adding the "else branch", it would have been better to change (the specification and) the implementation of the ceremony to forbid these mistakes (e.g., by programming the gates to warn the users that they are touching with the wrong card or with two cards), but we recognize that this might not always be possible, especially if all the software and hardware components of the System have already been deployed and installed.

If these matching mutations lead to an attack, then we can check with the ceremony designers whether the mutated specification makes sense and, in any case, use the obtained attack trace to generate concrete tests cases to be applied to the ceremony's implementation. This will allow us to check whether the attack entailed by the mutations is a false positive or a real attack.

The scenarios for the other human mutations and their matching mutations are similar. So, summarizing, our approach takes as input the specification of a ceremony and the goal(s) it should achieve, and then generates both mutations of the human agent's role (allowing him to add, skip or replace actions) and the matching mutations of the other roles of the ceremony. The resulting mutated ceremony specifications are then fed into Tamarin to search for attacks. We leave the step of concretizing the attack traces found into test cases as future work (although we expect this to be not too difficult by proceeding along the lines of [30], [39]).

4. Formal modeling of security ceremonies

We adopt, adapt and extend notions that are used in most of the state-of-the-art approaches and tools for the formal analysis of security protocols. For concreteness, our tool X-Men extends the Tamarin prover [6], [7], [26] to model and analyze security ceremonies with mutations caused by human users, but our approach is general and independent of Tamarin and could be applied similarly to other tools such as [3], [10], [19], [39]. We first summarize some basic notions (importing them from papers in which Tamarin is presented and used) and then discuss the formal specification of ceremonies, the execution model, the modeling of human agents, and the security goals.

4.1. Messages, ceremony specification and execution model

The *term algebra of messages* is given by $\mathcal{T}_\Sigma(\mathcal{V})$, where Σ is a signature and $\mathcal{V}$ is a disjoint, countably infinite set of variables. A term m is *ground* when it contains no variables. $F_{sym} \subset \Sigma$ denotes a finite set of function symbols that contains function symbols for: the *pairing* $pair(m_1, m_2)$ of two messages m_1 and m_2, also denoted by $\langle m_1, m_2 \rangle$, where, for brevity, we write, e.g., $\langle m_1, m_2, m_3 \rangle$ for $\langle m_1, \langle m_2, m_3 \rangle\rangle$; the *first projection* $\pi_1(m)$ and *second projection* $\pi_2(m)$ of a pair m of terms; the *hash* $h(m)$ of a term m; the *symmetric encryption* $senc(m, k)$ and the *symmetric decryption* $sdec(m, k)$ of m with k; the *asymmetric encryption* $aenc(m, k)$ and the *asymmetric decryption* $adec(m, k)$ of m with k; the *signature* $sign(m, k)$ and the corresponding *verification* $verify(sign(m, k_1), m, k_2)$. The function $pk(k)$ represents the public key corresponding to the private key k.

Messages are composed and decomposed using the standard Dolev-Yao-style equational theory for these functions. However, as we do for the Oyster ceremony, our approach allows us also not to consider explicitly the presence of a (Dolev-Yao) attacker and focus on capturing the way human agents might interact insecurely with the other ceremony agents. So, all our agents behave honestly and follow the steps of the ceremony, but the human(s) might make mistakes. In other cases, such as in the SSO ceremony (see § 5.2), we add an explicit attacker who intentionally tries to make the ceremony insecure.[4] In all these cases, our modeling of the human behavior (through mutations to the specification of the human(s) and of the agents the human(s) interact with) allows us to identify attacks that a standard Dolev-Yao attacker would not immediately be able to find.

Σ also contains a countably infinite set $\mathcal{C}_{fresh}$ of fresh constants, modeling the generation of nonces, and a countably infinite set $\mathcal{C}_{pub}$ of public constants, representing agent names and other publicly known values. The sets $F_{sym}, \mathcal{C}_{fresh}$ and $\mathcal{C}_{pub}$ are pairwise disjoint. We denote sequences with square brackets.

We say that m_1 is a *submessage* of m_2, in symbols $m_1 \in submsg(m_2)$, iff $m_2 = m_1$; $m_2 = \langle m_3, m_4 \rangle$ for some m_3, m_4 and $m_1 \in submsg(m_3)$ or $m_1 \in submsg(m_4)$; $m_2 = h(m_3)$ for some m_3 and $m_1 \in submsg(m_3)$; $m_2 = senc(m_3, k)$ for some m_3 and k and $m_1 \in submsg(m_3)$; $m_2 = aenc(m_3, k)$ for some m_3 and k and $m_1 \in submsg(m_3)$; or $m_2 = sign(m_3, k)$ for some m_3 and k and $m_1 \in submsg(m_3)$.

The *format* $f = format(m)$ of a message m is its top-level function symbol: if m has no top-level function symbol, then f is the identity function; if $m = \langle m_1, m_2 \rangle$ for some m_1, m_2, then $f = pair$; if $m = h(m_1)$ for some m_1, then $f = h$; if m is $\circ(m_1, k)$ for some m_1 and k, with $\circ \in \{senc, aenc, sign\}$, then $f = \circ$.

Formally, a *role script* is a sequence of events $e \in \mathcal{T}_{\Sigma \cup RoleActions}(\mathcal{V})$, where $RoleActions = \{Snd, Rcv, Start, Fresh\}$ and each event e has exactly one function symbol that is in $RoleActions$ at the top-level. We will introduce other Tamarin actions in our specifications (and simply write "actions" when there is no risk of confusion).

Send and *receive* events are of the form $Snd(A, l, P, m)$ and $Rcv(A, l, P, m)$, where A is the role executing the event, $l \in \{ins, auth, conf, sec\}$ indicates the type of channel over which a message is sent (insecure, authentic, confidential, secure), $P \in \mathcal{C}_{pub}$ is a role's name, and $m \in \mathcal{T}_\Sigma(V)$ is a message. In the $Snd(A, l, P, m)$ event, P is the intended recipient of the message m, whereas in $Rcv(A, l, P, m)$ event, P is the apparent sender, as the attacker may have forged the message, and m is the expected message pattern.

$Fresh(A, m)$ indicates that the role A generates a fresh message m (e.g., a nonce or a new key) and $Start(A, K)$ indicates the initial knowledge K of A. The start event is the first event of a role script and occurs only once.

As shown in Figure 3b, the Generalized Main Ceremony for the Tube has 3 roles: the human H and the entrance and exit gates $GateIn$ and $GateOut$. We remarked above that in this ceremony we do not consider cryptography (but we easily could) and, in fact, we do not consider an explicit attacker. We represent this by specifying that all messages are sent over secure channels.

4. We control the Dolev-Yao attacker by using (or not) appropriate channels. The messages used in the Oyster ceremony are not encrypted, but there is no reason why they could not be. The SSO ceremony, in contrast, includes explicit cryptographic operations.

Thus, the role scripts for the roles of this ceremony are:

$$
\begin{aligned}
&RoleScript_H = \\
&\quad [Start(H, \langle \text{`}GateIn\text{'}, \text{`}GateOut\text{'}, \text{`}card\text{'}\, \text{`}balance\text{'} \rangle \\
&\qquad\qquad \langle GateIn, GateOut, card, bal(card) \rangle\rangle), \\
&\quad Snd(H, \text{sec}, GateIn, \langle \text{`}card\text{'}, card \rangle), \\
&\quad Rcv(H, \text{sec}, GateIn, \langle \langle \text{`}card\text{'}, \text{`}gin\text{'} \rangle, \langle card, gin \rangle\rangle), \\
&\quad Snd(H, \text{sec}, GateOut, \langle \langle \text{`}card\text{'}, \text{`}balance\text{'}, \text{`}gin\text{'} \rangle, \\
&\qquad\qquad \langle card, bal(card), gin \rangle\rangle), \\
&\quad Rcv(H, \text{sec}, GateOut, \langle \langle \text{`}card\text{'}, \text{`}balance\text{'}, \text{`}finish\text{'} \rangle, \\
&\qquad\qquad \langle card, bal(card)', finish \rangle\rangle)] \\
&RoleScript_{GateIn} = \\
&\quad [Start(GateIn, \langle H, gin \rangle), \\
&\quad Rcv(GateIn, \text{sec}, H, \langle \text{`}card\text{'}, card \rangle), \\
&\quad Snd(GateIn, \text{sec}, H, \langle \langle \text{`}card\text{'}, \text{`}gin\text{'} \rangle, \langle card, gin \rangle\rangle)] \\
&RoleScript_{GateOut} = \\
&\quad [Start(GateOut, \langle H, gout \rangle), \\
&\quad Rcv(GateOut, \text{sec}, H, \langle \langle \text{`}card\text{'}, \text{`}balance\text{'}, \text{`}gin\text{'} \rangle, \\
&\qquad\qquad \langle card, bal(card), gin \rangle\rangle), \\
&\quad Snd(GateOut, \text{sec}, H, \langle \langle \text{`}card\text{'}, \text{`}balance\text{'}, \text{`}finish\text{'} \rangle, \\
&\qquad\qquad \langle card, bal(card)', finish \rangle\rangle)]
\end{aligned}
$$

We take advantage of *constants* in Tamarin to identify values received and sent during a ceremony. In [7], constants are used to define "tags" in order to represent the interpretation of the values in the knowledge of a human agent. We also make use of constants but we use them to define a basic notion of *types*. In this paper, as shown in the role scripts above and in the agent rules in Figure 8, we only consider types of ground terms, such as the type '$card$' for $card$ or '$balance$' for $bal(oyster)$.[5] This allows us to restrict what mutations can do, e.g., constants allow us to express that a payment card in a message is replaced with another card (instead of with a generic value that is not of type "card"). Still, for readability,

> *from now we will often omit constants in role scripts and rules, so that when you read m, please mentally replace it with the constant-message pair $\langle t, m \rangle$.*

Our approach is based on Tamarin's *execution model* [26], which is defined by a *multiset term-rewriting system* like in most other security protocol analysis tools. A *system state* is a multiset of *facts*: *linear facts* model exhaustible resources and they can be added to and removed from the system state, *persistent facts* model inexhaustible resources and can only be added to the system state (persistent fact symbols are prefixed with "!"). The *initial system state* is the empty multiset. A *trace* tr is a finite sequence of multisets of actions a and is generated by the application of labeled *state transition rules* of the form $prem \xrightarrow{a} conc$. Such a rule is applicable when the current state contains facts matching the premise $prem$, and the rule's application removes the matching linear facts from the state, adds instantiations of the facts in the conclusion $conc$ to the state, and records the instantiations of actions in a in the trace. The set of all traces of a set of rules $\mathcal{R}$ is denoted by $TR(\mathcal{R})$.

A *protocol model* consists of the agent rules, the fresh rule, channel rules and attacker rules. The *fresh rule* $[\,] \rightarrow [\mathsf{Fr}(x)]$ produces the fact $\mathsf{Fr}(x)$ where $x \in \mathcal{C}_{fresh}$; no two applications of the fresh rule pick the same element $x \in \mathcal{C}_{fresh}$ and this is the only rule that can produce terms $x \in \mathcal{C}_{fresh}$. Tamarin comes equipped with standard Dolev-Yao *attacker rules* and with *channel rules* (introduced in [6]) to model the sending and receiving of messages over authentic/confidential/secure channels, and thus control the ability of the attacker (who, e.g., can not send, read or replay messages on a secure channel, although he might still be able to interrupt the communication).

Agent rules specify the agents' state transitions and communication. For instance, the rules for the human agent in the Generalized Main Ceremony for the Tube are shown in Figure 8 (those for the other agents, which are similar, are in Appendix A). In general, for every event e in the script of a role A, we get a transition rule $prem \xrightarrow{a} conc$ as follows: the label of the rule contains the event, i.e., $e \in a$; $prem$ contains an agent state fact $\mathsf{AgSt}(A, step, kn)$, and $conc$ contains the subsequent agent state fact $\mathsf{AgSt}(A, step, kn')$, where $step$ refers to the role step the agent is in and kn is the agent's knowledge at that step. If $e \in a$ is: $Snd(A, l, P, m)$ then $conc$ additionally contains an outgoing message fact $\mathsf{Out}_l(A, P, m)$; $Rcv(A, l, P, m)$ then $prem$ contains an incoming message fact $\mathsf{In}_l(P, A, m)$; $Fresh(A, m)$ then $prem$ contains $\mathsf{Fr}(m)$; $Start(A, m)$ then it is translated to a setup rule where $conc$ contains the initial agent state $\mathsf{AgSt}(A, 0, m)$.[6]

As usual, the knowledge of an agent increases monotonically during the execution of the ceremony (as the agent receives messages or generates fresh terms).

4.2. Goals

Goals express the security properties that a ceremony is supposed to guarantee. However, many ceremonies, such as the Oyster ceremony as we discussed in §2, "push" security from the system to the human agents. This is made evident by the three goals that we define and analyze for the Oyster ceremony:

GO1 the human ends his journey touching in and out;
GO2 the human ends his journey using the same card to touch in and out;
GO3 the human does not touch two cards in and out.

These goals refer to a single journey, i.e., a single ceremony session. We formalize this in our Tamarin models by including explicit restrictions (through the *OnlyOnce* restriction [35] in the Setup phase; see our specifications in [40]) to force the human to carry out a single journey in the ceremony; given this, we can then formalize the goals as follows. Goal *GO1* can be formalized by the lemma

```
lemma complete_journey: all-traces
  "All H oyster #j. Hfin(H,'card',oyster)@j ==>
    (Ex GateIn gin #i. CommitGid(GateIn,H,gin) @i & i<j)"
```

which uses Tamarin *actions* to express that if H completes the ceremony with an Oyster card (action $Hfin(H, \text{`}card\text{'}, oyster)$) at time j, then there is a previous time instant i such that a $GateIn$ commits the gin to H (action $CommitGid(GateIn, H, gin)$). The other goals make use of other actions. For instance, a $Snd(A, l, P, m)$ event corresponds to the Tamarin action $Send(A, \langle t, m \rangle)$, which we abbreviate to $Send(A, m)$ following our readability assumption, whereas a $Rcv(A, l, P, m)$ event corresponds to the action $Receive(A, P, m)$ (note the absence of the tag t).

Goal *GO2* can be formalized by the lemma

5. This is enough for all ceremonies that we have encountered so far, so we leave a more thorough investigation of types to future work.

6. The translation of the different channels into Tamarin is quite natural, e.g., by means of rules such as $\mathsf{Out}_l(A, P, m) \rightarrow \mathsf{Out}(A, P, m)$ for $l \in \{ins, auth\}$ and $\mathsf{In}(P, A, m) \rightarrow \mathsf{In}_l(P, A, m)$ for $l \in \{ins, conf\}$.

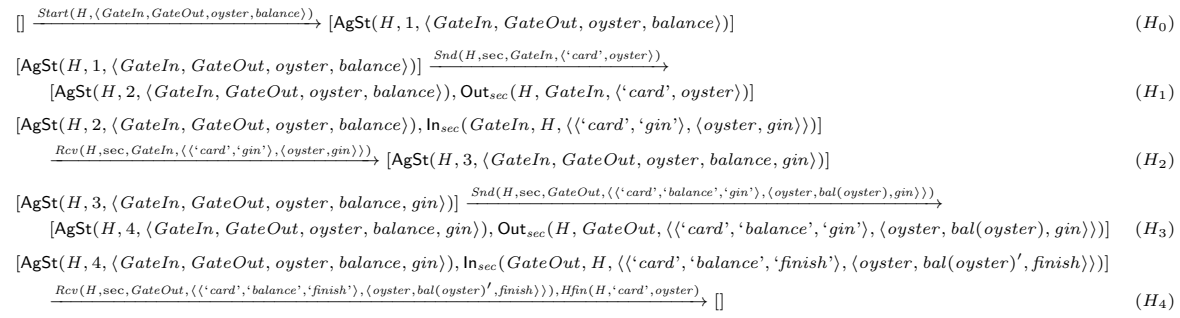


Figure 8: The rules for the human agent in the Generalized Main Ceremony for the Tube

```
lemma same_card: all-traces
  "All H oyster #j. Hfin(H,'card',oyster)@j
   ==> (Ex #t. Send(H,'card',oyster)@t & t<j)
        & not (Ex ccard #c. Send(H,'card',ccard)@c
                            & not (ccard = oyster))"
```

which expresses that if H completes the ceremony with an Oyster card ($Hfin(H, \text{`}card\text{'}, oyster)$) at time j, then H did not touch in another card.

Goal *GO3* can be formalized by the lemma

```
lemma Card_Clash_Out: all-traces
  "All H GateIn GateOut oyster gin #j #t.
   Receive(GateOut,H,oyster)@j
   & Commit(GateOut,H,'finish')@j
   & Receive(GateIn,H,oyster)@t
   & CommitGid(GateIn,H,gin)@t &  t<j
   & not (GateIn = GateOut)
   ==> not (Ex ccard #i #k. Receive(GateOut,H,ccard)@i
   & Commit(GateOut,H,'finish')@i
   & Receive(GateIn,H,ccard)@k
   & CommitGid(GateIn,H,gin)@k & k<i
   & not oyster = ccard)"
```

which expresses that if $GateOut$ receives $oyster$ from H ($Receive(GateOut, H, oyster)$) and commits the end of the journey ($Commit(GateOut, H, \text{`}finish\text{'})$) at time j, and a $GateIn$, which is not instantiated by the same agent as $GateOut$, receives $oyster$ from H and commits a gin ($CommitGid(GateIn, H, gin)$) to the same H at a previous t, then there does not exist another card $ccard$ such that the $GateOut$ and the $GateIn$ execute the same transitions receiving that $ccard$.

4.3. Modeling human mutations of the ceremony

The mutations that humans carry out when executing a ceremony have repercussions also on other agents and thus on the whole ceremony. We thus need to define not only the human mutations, which modify a ceremony trace by mutating the subtrace of the human agent, but also the mutations on the subtrace(s) of the other agent(s) that are likely (albeit not necessarily) required to "match" the mutations of the human; for instance, to receive the new or modified message sent by the human or to skip some actions mirroring the skip of the human. The result will be a fully mutated ceremony trace, which our tool feeds into Tamarin, first to check if it is executable and then to analyze it with respect to the corresponding goal(s).

Definition 1. A generic *human mutation* is a function $\mu^H : tr \to tr'$ that takes as input a trace tr and gives as output a new trace $tr' = [\![tr]\!]^{\mu^H}$ obtained by mutating H's subtrace as a consequence of the human H "deviating" from the original role script by skipping one or more actions, replacing a message with another one, or adding a new action.

A *matching mutation* for a human mutation is a mutation μ^m that mutates the subtraces of the other ceremony agents to match and propagate the human mutation.

The combination $\mu^H \circ \mu^m : tr \to tr'$ of the two mutations takes as input a trace tr and gives as output a new trace $tr' = [\![tr]\!]^{\mu^H \circ \mu^m}$ in which the human mutation is matched and propagated.

In the following subsections, we will instantiate these generic definitions to define the three human mutations *skip*, *replace* and *add* both formally and algorithmically, giving also the algorithmic definitions of the corresponding matching mutations. Slightly abusing notation, we will write $[a_0, \ldots, a_i, \ldots, a_n]^H$ with $0 < i \leqslant n$ to denote the subtrace of a human agent H in a ceremony execution, and let $[\![\_]\!]^\mu$ apply not just to traces. In fact, we consider mutations that apply generically to traces so that they apply indirectly also to role scripts and to Tamarin actions. For readability, and to make a clearer point, in the following descriptions and algorithms, we will sometimes depart from the tight corset of Tamarin's notation and consider transitions and their pre and postconditions. More specifically, in the style of multiset rewriting as in [33], we consider an abstract "merged" transition rule in which $prem$ contains the receipt of a message and $conc$ the sending of the reply:[7]

$$\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \to \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ Snd(H, l_2, A_2, m_2)\,,$$

where Pre_i is a set of precondition facts (e.g., fresh facts) at state i, $Post_{i+1}$ is a set of postcondition facts at state $i+1$, and kn_{i+1} is obtained by extending kn_i with m_1 and with whatever is generated fresh in Pre_1. As usual, kn_{i+1} is such that A can send the message m_2 (after closing the knowledge under the standard rules for message generation and analysis). It is not difficult to translate this transition to the two corresponding transition rules in Tamarin's notation (with In, Out, the Tamarin actions

7. This is in the spirit of the *step compression* technique that is adopted in several security protocol analysis tools, such as [3]. The idea is that some actions can be safely lumped together. For instance, we can safely assume that if a role is supposed to reply to a message it received, then we can compress the receive and send actions into a single transition.

and the constants) and vice versa, and to carry out the corresponding translations in the following descriptions and algorithms.

For instance, the following denotes a trace of the actions of a human agent H and possibly pairwise distinct agents $A_1, A_2, A_3, A_4, \ldots$

$$\begin{array}{c} \vdots\Sigma_1 \\ \mathsf{AgSt}(H,i,kn_i),\ Pre_i,\ Rcv(H,l_1,A_1,m_1) \rightarrow \\ \mathsf{AgSt}(H,i+1,kn_{i+1}),\ Post_{i+1},\ Snd(H,l_2,A_2,m_2) \\ \vdots\Sigma_2 \\ \mathsf{AgSt}(H,j,kn_j),\ Pre_j,\ Rcv(H,l_3,A_3,m_3) \rightarrow \\ \mathsf{AgSt}(H,j+1,kn_{j+1}),\ Post_{j+1},\ Snd(H,l_4,A_4,m_4) \\ \vdots\Sigma_3 \end{array} \quad (1)$$

where each Σ_i represents a possibly empty subtrace, and, as we wish to focus on H's actions, Σ_2 embeds A_2's receipt of m_2 in and Σ_3 embeds A_4's receipt of m_4. When $A_1 = A_2 = \ldots = A$, the trace reduces to a trace of a ping-pong ceremony between H and a system A.

4.3.1. The *skip* mutation.

Definition 2. A *skip* mutation $\mu^H_{skip} : tr \rightarrow tr'$ is a human mutation of tr's human subtrace $[a_0, \ldots, a_i, \ldots, a_n]^H$ such that tr' includes the new human subtrace $[a_0, \ldots, a_{i-1}, [\![a_{i+k}]\!]^\mu, \ldots, [\![a_n]\!]^\mu]$, where a_{i+k} with $k \geq 1$ is the action that H executes immediately after the execution of a_i and $[\![a_{i+k}]\!]^\mu, \ldots, [\![a_n]\!]^\mu$ are the mutations of these actions obtained by H skipping the actions $a_i, \ldots, a_{i+k-1}$ and by matching and propagating this mutation.

For example, Figure 9 shows a human subtrace in which H skips the $Snd(H, \mathrm{sec}, GateIn, card)$ action in the Oyster ceremony (omitting constants as discussed), which corresponds to not touching in. But this is not the only possible skip: H could skip also the receipt of the reply by $GateIn$ and jump to his next send to $GateOut$, which would actually make sense as one could argue that if $GateIn$ does not receive a message from H then it will not reply either; or H could skip both the receipt of a message and the sending of the reply; and so on.

We have identified five different *skip* mutations, depending on which send (S) and receive (R) actions are skipped: $\mu^H_{skip(S)}$, $\mu^H_{skip(SR)}$, $\mu^H_{skip(R)}$, $\mu^H_{skip(RS)}$ and $\mu^H_{skip(RSR)}$. More cases could be considered, but these five cover the most interesting scenarios, which can combined to skip bigger "chunks" of the ceremony execution.

We describe the five *skip* mutations by showing their effect on the subtrace (1).

The *skip* mutation $\mu^H_{skip(S)}$. In this case, H, having arrived at state $i+1$, skips the sending of m_2 and any other action that he would carry out in Σ_2 and continues the trace with the transition $j \geq i+1$, which we call the *landing transition* (i.e., the transition where H lands after the "jump" he has made):[8]

$$\begin{array}{c} \vdots\Sigma_1 \\ \mathsf{AgSt}(H,i,kn_i),\ Pre_i,\ Rcv(H,l_1,A_1,m_1) \rightarrow \\ \mathsf{AgSt}(H,i+1,kn_{i+1}),\ Post_{i+1},\ \cancel{Snd(H,l_2,A_2,m_2)} \\ \vdots[\![\Sigma_2]\!]^\mu \\ \mathsf{AgSt}(H,j,[\![kn_j]\!]^\mu),\ Pre_j,\ Rcv(H,l_3,A_3,[\![m_3]\!]^\mu) \rightarrow \\ \mathsf{AgSt}(H,j+1,[\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H,l_4,A_4,[\![m_4]\!]^\mu) \\ \vdots[\![\Sigma_3]\!]^\mu \end{array}$$

where μ denotes the mutation composed of $\mu^H_{skip(S)}$ and the matching and propagation entailed by $\mu^H_{skip(S)}$ (mutations of constant-message pairs are explained later).

This allows us to illustrate the need for matching and propagation on a concrete example. We namely need to consider if and how the mutated trace can be completed, for instance when H receives from A_3 an m_3 that is different from the expected one as a consequence of H's skipping the sending of m_2 to A_2. This immediately raises a number of questions. For instance, for $Rcv(H, l_3, A_3, [\![m_3]\!]^\mu)$ to be possible, it must be the case that $[\![\Sigma_2]\!]^\mu$ contains $Snd(A_3, l_3, H, [\![m_3]\!]^\mu)$, but there is no guarantee that this holds:

- if A_3 is able to send m_3 even when H does not send m_2 to A_2, then H can receive m_3, but
- if A_3 needs first A_2 (which is possibly but not necessarily equal to A_3) to receive m_2 to then be able to send m_3, then A_3 does not send m_3 in the mutated trace or sends a mutation of m_3 built from its current knowledge.

Our tool implements these options as described in the pseudo-code in Algorithm 1 and Algorithm 2.

The pseudo-code is hopefully quite explanatory, also thanks to the comments in the algorithms (whose start is denoted by ▷), but there are a couple of steps that deserve clarification. First of all, what does it mean that A_3 sends a mutation of m_3 built from its current knowledge? If we apply the message generation and analysis rules freely, this is an infinite set of possible messages. We could consider that as there is no guarantee of termination in our approach anyway, but instead we proceed in a more controlled way that mimics human users making mistakes when sending the messages or human programmers making mistakes when implementing a specification:

> *we consider only mutations of a message m that preserve the format of m.*

So, for example, in line 6 of Algorithm 1 we define $[\![m_3]\!]^\mu = \{(format(m_3))(m) \mid m \in submsg(m_3)\}$ of m_3, and then, for each of these mutations, we build all the corresponding transitions j (similarly, we build controlled mutations of m_4 in lines 6 and 13). Note also that we write $kn_l \cup Pre_l$ to mean the extension of kn_l with all messages generated freshly in Pre_l.

8. For simplicity but w.l.o.g., in the following we assume that the (fresh and "other") facts in Pre_j never refer to messages received during the execution of a ceremony, but only to long-term keys, public keys and the like; this entails that $[\![Pre_j]\!]^\mu = Pre_j$. This assumption allows us to avoid considering mutations of Pre_j induced by the situation in which a message is not received in $[\![\Sigma_2]\!]^\mu$. This is indeed the case in the Oyster and SSO examples. Extending our approach to capturing such mutations is cumbersome notationally but not difficult technically: we can define the mutation of the preconditions (and of the postconditions, if needed) in a way similar to the mutation of the knowledge when one or more messages are not received.

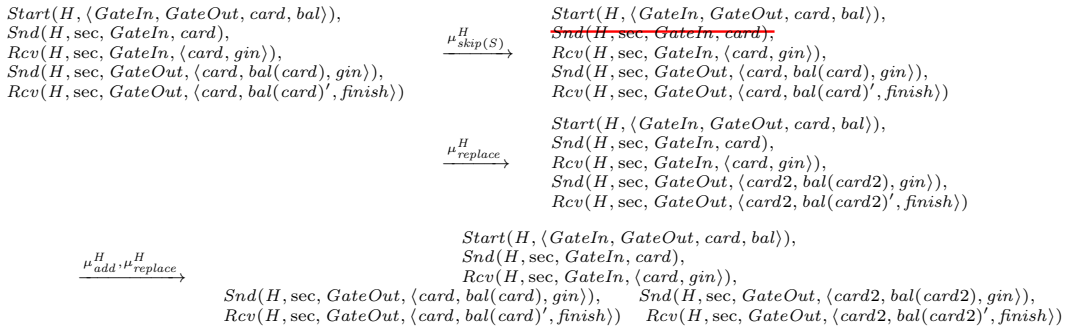


Figure 9: Examples of mutations in the case of the Oyster ceremony

The *skip* mutation $\mu^H_{skip(SR)}$. In this case, H skips $Snd(H, l_2, A_2, m_2)$ in transition i and $Rcv(H, l_3, A_3, m_3)$ in transition j:

$$\begin{array}{c}\vdots\Sigma_1\\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \rightarrow\\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ \cancel{Snd(H, l_2, A_2, m_2)}\\ \vdots[\![\Sigma_2]\!]^\mu\\ \mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j,\ \cancel{Rcv(H, l_3, A_3, [\![m_3]\!]^\mu)} \rightarrow\\ \mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)\\ \vdots[\![\Sigma_3]\!]^\mu\end{array}$$

The pseudo-code for this and the other cases of the *skip* mutation are given in Appendix B.

The *skip* mutation $\mu^H_{skip(R)}$. H skips $Rcv(H, l_1, A_1, m_1)$ in transition i:

$$\begin{array}{c}\vdots\Sigma_1\\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ \cancel{Rcv(H, l_1, A_1, m_1)} \rightarrow\\ \mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^\mu),\ Post_{i+1},\ Snd(H, l_2, A_2, [\![m_2]\!]^\mu)\\ \vdots[\![\Sigma_2]\!]^\mu\\ \mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j,\ Rcv(H, l_3, A_3, [\![m_3]\!]^\mu) \rightarrow\\ \mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)\\ \vdots[\![\Sigma_3]\!]^\mu\end{array}$$

The *skip* mutation $\mu^H_{skip(RS)}$. H skips both $Rcv(H, l_1, A_1, m_1)$ and $Snd(H, l_2, A_2, m_2)$ in transition i:

$$\begin{array}{c}\vdots\Sigma_1\\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ \cancel{Rcv(H, l_1, A_1, m_1)} \rightarrow\\ \mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^\mu),\ Post_{i+1},\ \cancel{Snd(H, l_2, A_2, [\![m_2]\!]^\mu)}\\ \vdots[\![\Sigma_2]\!]^\mu\\ \mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j,\ Rcv(H, l_3, A_3, [\![m_3]\!]^\mu) \rightarrow\\ \mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)\\ \vdots[\![\Sigma_3]\!]^\mu\end{array}$$

The *skip* mutation $\mu^H_{skip(RSR)}$. H skips both $Rcv(H, l_1, A_1, m_1)$ and $Snd(H, l_2, A_2, m_2)$ in transition i and $Rcv(H, l_3, A_3, m_3)$ in transition j:

$$\begin{array}{c}\vdots\Sigma_1\\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ \cancel{Rcv(H, l_1, A_1, m_1)} \rightarrow\\ \mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^\mu),\ Post_{i+1},\ \cancel{Snd(H, l_2, A_2, [\![m_2]\!]^\mu)}\\ \vdots[\![\Sigma_2]\!]^\mu\\ \mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j,\ \cancel{Rcv(H, l_3, A_3, [\![m_3]\!]^\mu)} \rightarrow\\ \mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)\\ \vdots[\![\Sigma_3]\!]^\mu\end{array}$$

Algorithm 1 $\mu^H_{skip(S)}$: skip $Snd(H, l_2, A_2, m_2)$ in transition i, with landing transition j as in (1)

1: **if** $[\![kn_j]\!]^\mu = kn_j = kn_{i+1}$ **then**
2: **if** $[\![\Sigma_2]\!]^\mu$ still contains a transition with $Snd(A_3, l_3, H, m_3)$ in its conclusions **then**
3: transition j is the same as the original one in trace tr, i.e.
4: $\mathsf{AgSt}(H, j, kn_j),\ Pre_j,\ Rcv(H, l_3, A_3, m_3) \rightarrow$ $\mathsf{AgSt}(H, j+1, kn_{j+1}),\ Post_{j+1},\ Snd(H, l_4, A_4, m_4)$
5: **else** ▷ $[\![\Sigma_2]\!]^\mu$ does not contain a transition with $Snd(A_3, l_3, H, m_3)$ in its conclusions
6: build all transitions j for all mutations $[\![m_3]\!]^\mu = \{(format(m_3))(m) \mid m \in submsg(m_3)\}$ of m_3 and for each of these, set $[\![kn_{j+1}]\!]^\mu = kn_j \cup \{[\![m_3]\!]^\mu\} \cup Pre_j$ and build all $[\![m_4]\!]^\mu = \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^\mu$, i.e.
7: $\mathsf{AgSt}(H, j, kn_j),\ Pre_j,\ Rcv(H, l_3, A_3, [\![m_3]\!]^\mu) \rightarrow$ $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
8: **else** ▷ this case is when $kn_j = kn_{i+1} \cup X$ for some set $X = \{M^x_1, ..., M^x_n\}$ of messages all different from m_3 and received by H in Σ_2
9: **if** $[\![\Sigma_2]\!]^\mu$ still contains a transition with $Snd(A_3, l_3, H, m_3)$ in its conclusions **then**
10: H skipped all transitions of Σ_2 in which he received $M^x_1, ..., M^x_n$,
11: $[\![kn_j]\!]^\mu = kn_j = kn_{i+1}$,
12: $[\![kn_{j+1}]\!]^\mu = kn_j \cup \{m_3\} \cup Pre_j$,
13: build all $[\![m_4]\!]^\mu = \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^\mu$, i.e.
14: $\mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j,\ Rcv(H, l_3, A_3, m_3) \rightarrow$ $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
15: **else** ▷ $[\![\Sigma_2]\!]^\mu$ does not contain a transition with $Snd(A_3, l_3, H, m_3)$ in its conclusions
16: H skipped all transitions of Σ_2 in which he received $M^x_1, ..., M^x_n$ and cannot receive m_3 in its original form but only in its mutated form $[\![m_3]\!]^\mu$
17: $[\![kn_j]\!]^\mu = kn_j = kn_{i+1}$,
18: **go to** 6

4.3.2. The *replace* mutation. This mutation captures the fact that a human user may send a message in place of another one (the case in which H replaces an action of a ceremony with another one is obtained by combining a *skip* and an *add* mutation).

If we were to allow the human to replace a message to send with any message that he can build out of his current knowledge, then we would have to deal with an infinite set of options (even if the human knows only one thing, such as his name, the message generation and analysis rules will allow him to generate an infinite set of messages); we will have the same problem with the *add* mutation that we consider next. In security protocol analysis, the ability of the attacker to generate infinitely many actors is a cause of non-termination of the analysis, which is controlled by considering only the messages that honest agents will actually respond to (and by introducing symbolic techniques, such as the "lazy intruder" [27],

Algorithm 2 Matching mutation for $\mu^H_{skip(S)}$

Consider the transition $next(i)$ that immediately follows the mutated human transition i
$\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
$\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$ where A_s is one of the other agents and l_p and m_p are some channel and message as specified in Σ_2.
remove $Rcv(A_2, l_2, H, m_2)$ from $next(i)$
$[\![kn_x]\!]^\mu = kn_{x-1}$
$[\![kn_{x+1}]\!]^\mu = kn_x \cup Pre_x$
build all $[\![m_p]\!]^\mu = \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^\mu$ i.e.
$\mathsf{AgSt}(A_2, x, [\![kn_x]\!]^\mu),\ Pre_x \rightarrow$
$\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^\mu),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^\mu)$
Let $h ::= next(i)$
if $\exists\, next(h)$ i.e.
  $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
  $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$
  then
    $[\![kn_s]\!]^\mu = kn_{s-1}$
    $m_p = [\![m_p]\!]^\mu$
    $[\![kn_{s+1}]\!]^\mu = [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$
    build all $[\![m_{p+1}]\!]^\mu = \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated by $[\![kn_{s+1}]\!]^\mu$ i.e.
    $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
    $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},$
    $Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
    go to 9 with $h ::= next(h)$

to manage the remaining infinite set of "answerable" messages). We cannot do that as our approach generates mutations in the behavior of the other agents to match the human mutations, so the other agents will be able to respond to any human message. Similar to [7], we thus restrict our attention to the messages that are already in the human's current knowledge kn (rather than in the knowledge's closure under the generation and analysis rules). More specifically, we consider $\mathcal{P}(kn) \setminus \emptyset$, the *powerset* of kn, i.e., the finite set of all subsets of kn including kn but excluding the empty set as it does not make sense to send an empty message. Moreover, to simplify further, we restrict our attention to messages of the same type. We leave the investigation of other controlled notions of "sendable" messages to future work.

Definition 3. A *replace* mutation $\mu^H_{replace} : tr \rightarrow tr'$ is a human mutation of tr's human subtrace $[a_0, \ldots, a_i, \ldots, a_n]^H$ such that tr' includes the new human subtrace $[a_0, \ldots, [\![a_i]\!]^\mu, [\![a_{i+1}]\!]^\mu, \ldots, [\![a_n]\!]^\mu]$, where a_i is a send action $Snd(H, l, A, m)$ and $[\![a_i]\!]^\mu$ is its mutation obtained by replacing the message m either with a sub-message (but preserving the format) or with a message contained in the powerset $\mathcal{P}(kn_i) \setminus \emptyset$ of H's current knowledge (but preserving types as specified by the corresponding constants); $[\![a_{i+1}]\!]^\mu, \ldots, [\![a_n]\!]^\mu$ are the mutations of the actions $a_{i+1}, \ldots, a_n$ obtained by matching and propagating this mutation.

Again, we show the effect of the $\mu^H_{replace}$ mutation by showing its effect on the subtrace (1):

$$\begin{array}{c} \vdots_{\Sigma_1} \\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \rightarrow \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ Snd(H, l_2, A_2, [\![m_2]\!]^\mu) \\ \vdots_{[\![\Sigma_2]\!]^\mu} \\ \mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^\mu),\ Pre_{i+1},\ Rcv(H, l_3, A_3, [\![m_3]\!]^\mu) \rightarrow \\ \mathsf{AgSt}(H, i+2, [\![kn_{i+2}]\!]^\mu),\ Post_{i+2},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu) \\ \vdots_{[\![\Sigma_3]\!]^\mu} \end{array}$$

where μ, which denotes the mutation composed of $\mu^H_{replace}$ and the matching and propagation entailed by $\mu^H_{replace}$, replaces m_2 either with $[\![m_2]\!]^\mu = \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ or with a message m that has the same constant as m_2 and is obtained from H's current knowledge as shown in Algorithm 11, which is given in Appendix C along with Algorithm 12 for the matching mutation.[9]

For example, H could start the Oyster ceremony with one card $card1$ and complete it with another card $card2$, thus giving rise to two "incomplete journeys", as shown in Figure 9 and discussed in §5.1. For another example, see the SSO ceremony in §5.2.

4.3.3. The *add* mutation. There are two different cases for this mutation: the human could

- send at any time any message that is in the powerset of the messages that are in his current knowledge, or
- duplicate a send action.[10]

Definition 4. An *add* mutation is a mutation $\mu^H_{add} : tr \rightarrow tr \times tr'$ such that the original trace $tr = [a_0, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n]$ is run in parallel with the new, mutated trace $tr' = [a_0, \ldots, a_{i-1}, [\![a_i]\!]^\mu, [\![a_{i+1}]\!]^\mu, \ldots, [\![a_n]\!]^\mu]$, where a_i is a send action and $[\![a_i]\!]^\mu$ is its possible mutation obtained either by duplicating a_i or by adding an action $Snd(H, l, A, m)$ at state i for some l with $A, m \in \mathcal{P}(kn_i) \setminus \emptyset$, and $[\![a_{i+1}]\!]^\mu, \ldots, [\![a_n]\!]^\mu$ are the mutations of the actions $a_i, \ldots, a_{i+k-1}$ obtained by matching and propagating this mutation.

Consider the beginning of the subtrace (1). μ^H_{add} mutates this to either

$$\begin{array}{c} \vdots_{\Sigma_1} \\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, M_1) \rightarrow \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ Snd(H, l_2, A_2, M_2) \\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, M_1) \rightarrow \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Snd(H, l, A, M) \\ \vdots_{[\![\Sigma_2]\!]^\mu} \end{array}$$

or

$$\begin{array}{c} \vdots_{\Sigma_1} \\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, M_1) \rightarrow \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ Snd(H, l_2, A_2, M_2) \\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, M_1) \rightarrow \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Snd(H, l_2, A_2, [\![M_2]\!]^\mu) \\ \vdots_{[\![\Sigma_2]\!]^\mu} \end{array}$$

The add mutation is best exemplified when combined with the other mutations, e.g.,

- sending at any time any message that he knows can be combined with a *skip* mutation (e.g., to skip some steps of a ceremony and instead send

9. We give only one algorithm for $\mu^H_{replace}$ since, differently from the cases of the *skip* mutations, the two subcases of $\mu^H_{replace}$ proceed in the same way after the replacement of m_2.

10. Note that we only consider mutations initiated by a human agent; as a consequence, we do not consider the situation in which the human agent initiates a mutation of the ceremony by adding a receive action as that would require another agent (human or not) to have added the corresponding send action first.

Mutation		Attacker	Case Study	
			Oyster	SSO
skip	S	-	2	-
	SR	-	3	-
	R	-	2	-
	RS	-	1	-
	RSR	-	1	-
replace	submessage	✓	-	232
	type	-	2	-
add		-	92	
add&replace	submessage	✓	-	232
	type	-	2	-

Table 1: Mutated models generated by X-Men

an arbitrary message before continuing with the rest of the ceremony),

- duplicating a send action can be combined with a *replace* mutation (as we do in our case studies).

For example, H could start the Oyster ceremony touching in with one card $card1$ and then, by mistake, touch out with two cards; this can be represented by adding a second touch-out send message where the first card is replaced with the second, thus obtaining the two traces shown in Figure 9.

Our tool implements this mutation as described in the pseudo-code in Algorithm 13, which is given in Appendix D along with Algorithm 14 for the matching mutation.

5. X-Men: a tool for the generation of mutated models based on human behaviors

In this section, we show how our formalization can be used effectively for discovering attacks that are due to the behavior of human agents in security ceremonies. As proof-of-concept, we have applied our tool X-Men to two case studies, the Oyster ceremony and the Single Sign-On ceremony. As explained in more detail in the following, X-Men generated a large number of mutated models for these ceremonies, which we have analyzed using Tamarin.

X-Men is a prototype tool written in Java. As shown in Figure 1, X-Men takes as input a model of the security ceremony (a specification file in the .spthy format, where .spthy stands for security protocol theory) and executes a Python script that splits the model into channel rules, agent rules and all other rules and the goals. The security analyst employing X-Men then selects the desired mutation (the three we have defined here or their combinations, and any other mutation that will be defined in X-Men's library of behavioral patterns in the future) to mutate the agent and the other rules, which are then merged with the original channel rules and goals to produce the many different mutated models that can be input to Tamarin.

X-Men generated a large number of mutated models for our two case studies, as shown in Table 1. We then used Tamarin to analyze individually these mutated models. Let us now summarize the results of the analysis (we lack space to discuss them in detail).

5.1. Analysis of the Oyster ceremony

Table 2 shows some of the attacks found with the models obtained using the mutations applied to the Oyster ceremony (due to lack of space, we exclude the results pertaining to the 92 models generated by the add mutation). "Mutated model" lists the file identifier of the generated file (as used at [40]) and the table also provides mutation details as well as a brief explanation of the human agent's behavior for each model. In addition to the three goals discussed in § 4.2, we have used Tamarin to check the *functional* goal (a.k.a. *executable* goal) that the mutations did not create models in which the legitimate execution trace of the ceremony is not valid anymore. All the models considered in the table passed this check.

We describe three interesting attacks that Tamarin has been able to discover out of the many mutations generated.

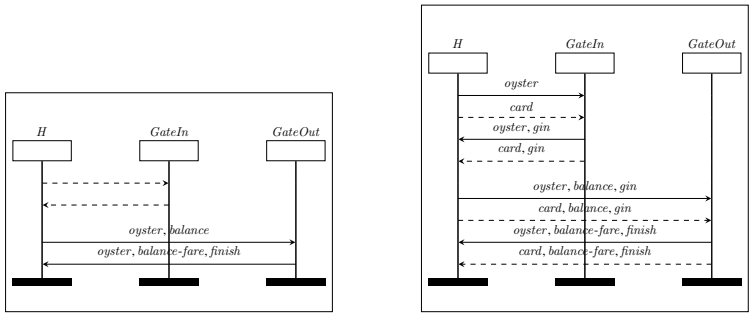
Attack #1. The MSC of the attack (Figure 10a) shows how the human agent H may execute the Oyster ceremony without touching-in at the entrance (as shown by the dotted arrows representing the human agent who is not touching-in). H touches-out the $oyster$ and $GateOut$ reads the information saved on the card, which does not specify where H entered as $GateIn$ was not able to write its identifier gin on the card. The security goal *GO1* is not verified, entailing what TfL calls an *incomplete journey* as mentioned in § 2, and the system charges a penalty fare as it is not able to calculate the journey of the passenger.

This is a real scenario that occurs when the passenger forgets to touch-in, e.g., when the station has no proper gates but only card readers at the station entrance, when the gates are already open (TfL sometimes opens the gates to speed up entry/exit during rush hour or when there are a large number of passengers), or when the reader is not working properly and does not read/write the Oyster card.

Attack #2. H may use two different cards in a single journey, touching-in with the first and touching-out with the second, so that *GO2* fails with two incomplete journeys. This may appear to be an uncommon scenario, but several tourists and even Londoners suffered from this problem. For instance, the passenger might have two Oyster cards in their pockets and confuse them, or the passenger might use Apple/Google pay (cf. § 2) but using two different devices (say smartphone and smartwatch), which will cause two incomplete journeys because the *Device Account Number* is unique for each device and is used by TfL as the identifier for a single journey.

Attack #3. The MSC in Figure 10b shows how H may use two cards (e.g., Oyster and a contactless card), simultaneously touching them both in/out when entering/exiting (as shown by the dotted arrows representing a parallel second execution of the Oyster ceremony), so that *GO3* fails due to a *card clash* (cf. § 2). This occurs, e.g., when a passenger touches with a wallet that holds all the passenger's cards that the system considers to be valid payment cards.

The attacks on the Oyster ceremony were discovered using the mutations generated by X-Men as shown in Table 2. The analysis did not require the activation of a Dolev-Yao attacker as the system, through the matching mutations, replied and billed the passengers "wrongly" due to their mistakes. While these attacks are, to some extent, known to TfL (cf. their warnings in Figure 4) and can be gathered empirically by observing the concrete behavior of the Tube passengers, it is important to stress that X-Men allows us to discover them automatically. Other attacks might be discovered by considering other



(a) Incomplete journey scenario (b) Card clash scenario

Figure 10: Two attack scenarios for the Oyster ceremony

Mutation		Mutated model	Mutation details	Explanation	Goal: complete journey (*GO1*)	Goal: same card (*GO2*)	Goal: card clash (*GO3*)	Goal: functional
skip	S	M0	Skip send in H_1	H does not touch in	✓	✓	×	•
		M1	Skip send in H_2	H does not touch out	×	×	×	•
	SR	M0	Skip send in H_1 and receive in H_2	H does not touch in (here H ignores any response from $GateIn$)	✓	×	×	•
		M1	Skip send in H_1 and receive in H_3	H does not touch in (here H could receive a response from $GateIn$ but ignores any response from $GateOut$)	✓	✓	×	•
		M2	Skip send in H_2 and receive in H_3	H does not touch out (here H ignores any response from $GateOut$)	×	×	×	•
	R	M0	Skip receive in H_2	H does not receive confirmation of touch in	✓	×	×	•
		M1	Skip receive in H_3	H does not receive confirmation of touch out	×	×	×	•
	RS	M0	Skip receive and send in H_2 and receive in H_3	H does not receive confirmation of touch in and does not touch out (here H could receive a response from $GateOut$)	×	×	×	•
	RSR	M0	Skip receive and send in H_2 and receive in H_3	H does not receive confirmation of touch in and does not touch out (here H ignores any response from $GateOut$)	✓	×	×	•
replace		M0	Replace *oyster* with *ccard* in whole ceremony	H uses a contactless card instead of Oyster	×	×	×	•
		M1	Replace *oyster* with *ccard* only in H_2 and after	H touches out using a different card	×	✓	×	•
		M2	Replace $bal(oyster)$ with $bal(ccard)$	H uses balance of ccard instead of Oyster	×	×	×	•
add&replace		M0	Similar to "replace M0" keeping the original ceremony	H uses a contactless card instead of Oyster	×	✓	✓	•
		M1	Similar to "replace M1" keeping the original ceremony	H touches out using a different card	×	✓	×	•
		M2	Similar to "replace M2" keeping the original ceremony	H uses balance of ccard instead of Oyster	×	×	×	•

Table 2: Some of the attacks found on the models obtained using the mutations applied to the Oyster ceremony (✓ indicates that an attack has been found, × indicates that no attack was found, • indicates that the functional goal is verified)

goals or other mutations. Moreover, in the style of model-based testing (see the end of § 3), it is possible to use the attack traces to generate concrete test cases to be executed on the code of the ceremony (if that is available).

To provide an example of the analysis in the presence of an attacker, we have considered the SSO ceremony.

5.2. Analysis of the SSO ceremony

The *SAML-based Single Sign-on for Google Apps* suffered from a well-know man-in-the-middle attack [4]. The protocol relies on the use of an authentication assertion $AuthAssert(ID, C, IdP, SP)$ signed by the identity provider IdP, where ID is an identifier and SP is the provider of a service that a client C wishes to use. In a nutshell (see [4] for details), the attack was due to the fact that the implementation had simplified $AuthAssert(ID, C, IdP, SP)$ into $AuthAssert(C, SP)$ to speed up the digital signature. A malicious SP can use this assertion to pose as C in another run of the protocol.

We have specified this SSO in X-Men, considering what would happen if SP is played by the attacker and IdP is played by a human, who may mistakenly generate and sign a wrong authentication assertion. Indeed, the *replace (submessage)* mutation generates $AuthAssert(C, SP)$ among other mutations. We have formalized the goal "IdP authenticates only the agent who requires to be authenticated" as a standard injective-agreement goal in Tamarin (see Appendix E) and indeed we were able to find the attack. This shows that our approach is able to find an attack that was not present in the original specification of SAML-based Single Sign-on but was introduced in Google's implementation [4]. Our mutations, among other things, capture such possible specification-implementation deviances.

6. Related work

In [28], Paulson introduced the *Oops* rule to model mistakes done by agents when executing a security protocol, such as the loss (by any means) of a session key. However, the notion of security ceremony and the explicit investigation of the consequences of explicitly considering

human agents and their mistakes was introduced by Ellison in [18], one of the pioneers of *socio-technical security*.

One of the first formal approaches to investigate security ceremonies is the *concertina* model introduced in [9], which spans over a number of socio-technical layers, focusing in particular on the socio-technical protocol between a user persona and a computer interface, but without explicitly considering human mistakes nor accounting for an explicit attacker. Similarly, the approaches in [12], [25] provide a formal model to reason about how a Dolev-Yao-style attacker can attack the communication between humans and computers, including storing of human knowledge, but without explicitly considering human mistakes.

In contrast, Basin et al. [7] provide a formal model for reasoning about some errors that humans involved in security protocols may make. They specify rules formalizing different types of humans (untrained, infallible or fallible humans), modeling a human who can send and receive any messages, resulting in attacks because a human discloses information, but also in attacks because the human just enters the same information on the wrong device or accepts a received message he should not. They successfully applied their model to analyze some authentication protocols. Although their approach is similar in spirit to ours, there are some fundamental differences along with some affinities. The two main differences are the following ones. First, they only consider scenarios in which the Dolev-Yao attacker actively attacks the protocol, whereas our approach works also when the attacker is not present thanks to the matching mutations. Second, similar to what we do, they also consider an add rule that allows humans to send "controlled" messages that are in their current knowledge, but our mutations allow us to capture a different, and to some extent wider, set of human deviations from the original ceremony.

Similar to [7], Curzon et al. [14] propose a formal human model that includes a specific attacker able to exploit the errors against the human user. The errors considered are those caused by the humans' interpretation of the system and by the design of the interfaces, but not those entailed by human choices or mistakes as we do. Moreover, they do not consider communication channels.

Johansen and Jøsang [23] define probabilistic processes to model the actions of a human agent, separating the model of the human and that of the user interface. They introduce a "compilation" operation in order to capture the interaction of the human agent and the user interface. Their probabilistic model for the human agent is an extension of the *persona model* [34]. Their approach provides only a preliminary formalization without a security analysis.

Beckert and Beuster [8] provide a formal semantics for GOMS models augmented with formal models of the application and the user's assumptions about the application, but they do not consider human mistakes in detail.

Pavlovic and Meadows [29] employ *actor-networks* as a formal model of computation and communication in networks of computers, humans and their devices, but they too do not consider human mistakes in detail.

Radke and Boyd [31] introduce the notion of *human-followable security* wherein a human user can understand the process and logic behind authentication protocols. They focus on showing how to transform existing authentication protocols into protocols with human-followable security.

While our approach is quite radically different from the research in [8], [14], [23], [29], [31], we believe that there might be interesting synergies between our mutations and the way in which they model the assumptions and perceptions of the human users, which we plan to investigate in future work.

7. Conclusions

Our approach allows security analysts to consider human "shades of gray" in the analysis of security ceremonies. We have already mentioned a number of directions for future work. We also plan to: extend the current mutations by weakening some of the constraints (e.g., on the types and formats of the messages, say to consider other controlled notions of "sendable" message or the case in which the right message is composed in a wrong format); consider other abilities of the attacker (e.g., as in [5]); extend X-Men's library of behavioral patterns with other mutations; formalize combinations of mutations and prove compositionality results; improve the efficiency of our approach by reducing the number of generated mutated models (e.g., by identifying isomorphic models) and by automatically checking whether attacks are real or not; link our formal analysis to mutation testing by generating test cases out of the attack traces; and, finally, consider other, even more complex, case studies.

Acknowledgment

We thank Giampaolo Bella, Rosario Giustolisi, Gabriele Lenzini, Sebastian Mödersheim, Ralf Sasse and the anonymous reviewers for their useful comments and suggestions.

References

[1] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. *SIGPLAN Not.*, (3):104–115, 2001.

[2] O. Almousa, S. Mödersheim, and L. Viganò. Alice and Bob: Reconciling Formal Models and Implementation. In *Programming Languages with Applications to Biology and Security*, LNCS 9465, pages 66–85. Springer, 2015.

[3] A. Armando, W. Arsac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuéllar, G. Erzse, S. Frau, M. Minea, S. Mödersheim, D. von Oheimb, G. Pellegrino, S. Ponta, M. Rocchetto, M. Rusinowitch, M. Torabi Dashti, M. Turuani, and L. Viganò. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In *TACAS*, LNCS 7214, pages 267–282. Springer, 2012.

[4] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-on for Google Apps. In *FMSE*, pages 1–10. ACM Press, 2008.

[5] M. Backes, J. Dreier, S. Kremer, and R. Künnemann. A Novel Approach for Reasoning about Liveness in Cryptographic Protocols and Its Application to Fair Exchange. In *EuroS&P 2017*, pages 76–91. IEEE, 2017.

[6] D. A. Basin, S. Radomirovic, and M. Schläpfer. A Complete Characterization of Secure Human-Server Communication. In *Computer Security Foundations Symposium (CSF)*, pages 199–213. IEEE, 2015.

[7] D. A. Basin, S. Radomirovic, and L. Schmid. Modeling Human Errors in Security Protocols. In *Computer Security Foundations Symposium (CSF)*, pages 325–340. IEEE, 2016.

[8] B. Beckert and G. Beuster. A method for formalizing, analyzing, and verifying secure user interfaces. In *International Conference on Formal Engineering Methods*, pages 55–73. Springer, 2006.

[9] G. Bella and L. Coles-Kemp. Layered analysis of security ceremonies. *IFIP Advances in Information and Communication Technology*, pages 273–286, 2012.

[10] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW*. IEEE, 2001.

[11] M. Büchler, J. Oudinet, and A. Pretschner. Security Mutants for Property-Based Testing. In *Tests and Proofs*, LNCS 6706, pages 69–77. Springer, 2011.

[12] M. C. Carlos, J. E. Martina, G. Price, and R. F. Custódio. A proposed framework for analysing security ceremonies. In *SECRYPT*, pages 440–445. Scitepress Digital Library, 2012.

[13] N. Courtois, K. Nohl, and S. O'Neil. Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards. *IACR Cryptology ePrint Archive*, page 166, 2008.

[14] P. Curzon, R. Rukšėnas, and A. Blandford. An approach to formal verification of human–computer interaction. *Formal Aspects of Computing*, 19(4):513–550.

[15] F. Dadeau, P.-C. Héam, R. Kheddam, G. Maatoug, and M. Rusinowitch. Model-based mutation testing from security protocols in hlpsl. *Software Testing, Verification and Reliability*, 25(5-7):684–711, 2015.

[16] G. de Koning Gans, J.-H. Hoepman, and F. D. Garcia. A practical attack on the MiFare Classic. In *CARDIS*, LNCS 5189, pages 267–282. Springer, 2008.

[17] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Program Mutation: A New Approach to Program Testing. *Infotech State of the Art Report, Software Testing*, 1979.

[18] C. M. Ellison. Ceremony Design and Analysis. *IACR Cryptology ePrint Archive*, pages 1–17, 2007.

[19] S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V. FOSAD 2009, FOSAD 2007, FOSAD 2008*, LNCS 5705, pages 1–50. Springer, 2009.

[20] T. Fábrega, F. Javier, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of computer security*, (2-3):191–230, 1999.

[21] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. Van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling MiFare Classic. In *ESORICS*, LNCS 5283, pages 97–114. Springer, 2008.

[22] Y. Jia and M. Harman. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[23] C. Johansen and A. Jøsang. Probabilistic modelling of humans in security ceremonies. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance (DPM, QASA, SETOP)*, LNCS 8872, pages 277–292. Springer, 2014.

[24] J. E. Martina, E. dos Santos, M. C. Carlos, G. Price, and R. F. Custódio. An adaptive threat model for security ceremonies. *International Journal of Information Security*, (2):103–121, 2015.

[25] J. E. Martina, T. C. Salavaro de Souza, and R. F. Custódio. Ceremonies Formal Analysis in PKI's Context. In *Computational Science and Engineering (CSE)*, pages 392–398. IEEE, 2009.

[26] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin. The Tamarin prover for the symbolic analysis of security protocols. In *CAV 2013*, LNCS 8044, pages 696–701. Springer, 2013.

[27] S. Mödersheim and L. Viganò. The Open-Source Fixed-Point Model Checker for Symbolic Analysis of Security Protocols. In *Foundations of Security Analysis and Design V. FOSAD 2009, FOSAD 2007, FOSAD 2008*, LNCS 5705, pages 166–194. Springer, 2009.

[28] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of computer security*, (1-2):85–128, 1998.

[29] D. Pavlovic and C. Meadows. Actor-Network Procedures (Extended Abstract). In *ICDCIT*, LNCS 7154, pages 7–26. Springer, 2012.

[30] M. Peroli, F. De Meo, L. Viganò, and D. Guardini. MobSTer: A Model-Based Security Testing Framework for Web Applications. *Software Testing, Verification & Reliability*, 28(8), 2018.

[31] K. Radke and C. Boyd. Security Proofs for Protocols Involving Humans. *Comput. J.*, 60(4):527–540, 2017.

[32] K. Radke, C. Boyd, J. M. Gonzalez Nieto, and M. Brereton. Ceremony Analysis: Strengths and Weaknesses. In *SEC*, IFIP AICT 354, pages 104–115. Springer, 2011.

[33] B. Schmidt, S. Meier, C. Cremers, and D. A. Basin. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *Computer Security Foundations Symposium (CSF)*, pages 78–94. IEEE, 2012.

[34] R. Semančík. Basic properties of the persona model. *Computing and Informatics*, 26(2):105–121, 2007.

[35] The Tamarin User Manual. https://tamarin-prover.github.io.

[36] TfL (Transport for London). Card Clash.

[37] TfL (Transport for London). Incomplete Journeys.

[38] TfL (Transport for London). TfL Transparency Strategy.

[39] L. Viganò. The SPaCIoS project: Secure Provision and Consumption in the Internet of Services. In *ICST*, pages 497–498. IEEE, 2013.

[40] X-Men: A Mutation-Based Approach for the Formal Analysis of Security Ceremonies. https://sempreboni.github.io/X-Men/.

Appendix

1. Agent rules

The rules for the $GateIn$ and $GateOut$ agents in the Generalized Main Ceremony for the Tube are given in Figure 11 and Figure 12, respectively.

2. The other four cases of the *skip* mutation and the other matching mutations for the *skip* mutation

Algorithm 3, Algorithm 4, Algorithm 5 and Algorithm 6 show the other four cases of the *skip* mutation.

Algorithm 7, Algorithm 8, Algorithm 9 and Algorithm 10 show the other matching mutations for the *skip* mutation.

Algorithm 3 $\mu^H_{skip(SR)}$: skip $Snd(H, l_2, A_2, m_2)$ in i and $Rcv(H, l_3, A_3, m_3)$ in j

1: **if** $[\![kn_j]\!]^\mu = kn_j = kn_{i+1}$ **then**
2: $\quad [\![kn_{j+1}]\!]^\mu = kn_j \cup Pre_j$,
3: $\quad$ build all $[\![m_4]\!]^\mu = \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^\mu$, i.e.
4: $\quad \mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j,\ \rightarrow$
$\quad \mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
5: **else** ▷ this case is when $kn_j = kn_{i+1} \cup X$ for some set $X = \{M^x_1, ..., M^x_n\}$ of messages all different from m_3 and received by H in Σ_2
6: $\quad H$ skipped all transitions of Σ_2 in which he received $M^x_1, ..., M^x_n$,
7: $\quad [\![kn_j]\!]^\mu = kn_j = kn_{i+1}$,
8: $\quad$ **go to** 1

3. The *replace* and *add* mutations

Algorithm 11 shows the *replace* mutation and Algorithm 12 shows the matching mutation.

$$[] \xrightarrow{Start(GateIn,\langle H,gin\rangle)} [\mathsf{AgSt}(GateIn, 1, \langle H, gin\rangle)] \quad (Gi_0)$$

$$[\mathsf{AgSt}(GateIn, 1, \langle H, gin\rangle), \mathsf{In}_{sec}(H, GateIn, \langle `card', oyster\rangle)] \xrightarrow{Rcv(GateIn,\mathrm{sec},H,\langle `card', oyster\rangle)} [\mathsf{AgSt}(GateIn, 2, \langle H, gin, oyster\rangle)] \quad (Gi_1)$$

$$[\mathsf{AgSt}(GateIn, 2, \langle H, gin, oyster\rangle)] \xrightarrow{Snd(GateIn,\mathrm{sec},H,\langle\langle `card', `gin'\rangle\langle oyster, gin\rangle\rangle),CommitGid(GateIn,H,gin)} [\mathsf{Out}_{sec}(GateIn, H, \langle\langle `card', `gin'\rangle, \langle oyster, gin\rangle\rangle)] \quad (Gi_2)$$

Figure 11: Agent rules for the $GateIn$ in the Generalized Main Ceremony for the Tube

$$[] \xrightarrow{Start(GateOut,\langle H,gout\rangle)} [\mathsf{AgSt}(GateOut, 1, \langle H, gout\rangle)] \quad (Go_0)$$

$$[\mathsf{AgSt}(GateOut, 1, \langle H, gout\rangle), \mathsf{In}_{sec}(H, GateOut, \langle\langle `card', `balance', `gin'\rangle, \langle oyster, bal(oyster), gin\rangle\rangle)]$$
$$\xrightarrow{Rcv(GateOut,\mathrm{sec},H,\langle\langle `card', `balance', `gin'\rangle, \langle oyster, bal(oyster), gin\rangle\rangle)} [\mathsf{AgSt}(GateOut, 2, \langle H, GateOut, oyster, bal(oyster), gin\rangle] \quad (Go_1)$$

$$[\mathsf{AgSt}(GateOut, 2, \langle H, GateOut, oyster, bal(oyster), gin\rangle]$$
$$\xrightarrow{Snd(GateOut,\mathrm{sec},H,\langle\langle `card', `balance', `finish'\rangle\langle oyster, bal(oyster)', `finish'\rangle\rangle),Commit(GateOut,H,`finish')}$$
$$[\mathsf{Out}_{sec}(GateOut, H, \langle\langle `card', `balance', `finish'\rangle, \langle oyster, bal(oyster)', `finish'\rangle\rangle)] \quad (Go_2)$$

Figure 12: Agent rules for the $GateOut$ in the Generalized Main Ceremony for the Tube

Algorithm 4 $\mu^H_{skip(R)}$: skip $Rcv(H, l_1, A_1, m_1)$ in transition i

1: **if** $[\![kn_{i+1}]\!]^\mu = kn_{i+1} = kn_i \cup Pre_j$ **then**
2: transition i is the same as the original one in trace t, without the $Rcv(H, l_1, A_1, m_1)$, i.e.
3: $\mathsf{AgSt}(H, i, kn_i)$, $Pre_i \rightarrow$ $\mathsf{AgSt}(H, i+1, kn_{i+1})$, $Post_{i+1}$, $Snd(H, l_2, A_2, m_2)$
4: **else** ▷ this case is when $kn_{i+1} = kn_i \cup X \cup Pre_i$, that means that m_1 has new knowledge
5: $[\![kn_{i+1}]\!]^\mu = kn_i \cup Pre_i$,
6: build all $[\![m_2]\!]^\mu = \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ that can be generated by $[\![kn_{i+1}]\!]^\mu$, i.e.
7: $\mathsf{AgSt}(H, i, kn_i)$, $Pre_i \rightarrow$ $\mathsf{AgSt}(A, i+1, [\![kn_{i+1}]\!]^\mu)$, $Post_{i+1}$, $Snd(H, l_2, A_2, [\![m_2]\!]^\mu)$

4. The *add* mutations

Algorithm 13 shows the *add* mutation and Algorithm 14 shows the matching mutation.

5. The specifications of the SSO ceremony

The goal of the Single Sign-On ceremony

> The Identity Provider IdP authenticates only the agent who requires to be authenticated.

can be formalized in Tamarin by a standard injective-agreement lemma:

```
lemma injective_agree:
  "All actor peer params #i.
       Commit(actor, peer, params)@ i
       ==>
        (Ex #j. Running(actor, peer, params)@ j & j < i
         & not(Ex actor2 peer2 #i2.
                Commit(actor2, peer2, params)@ i2
                & not(#i = #i2)))
       | (Ex #r. RevLtk(actor) @ r)
       | (Ex #r. RevLtk(peer)  @ r)"
```

Algorithm 5 $\mu^H_{skip(RS)}$: skip $Rcv(H, l_1, A_1, m_1)$ and $Snd(H, A_2, l_2, m_2)$ in transition i, with landing transition j

1: **if** $[\![kn_j]\!]^\mu = kn_j = kn_i$ **then**
2: **if** $[\![\Sigma_2]\!]^\mu$ still contains a transition with $Snd(A_3, l_3, H, m_3)$ in its RHS **then**
3: transition j is the same as the original one in trace t, i.e.
4: $\mathsf{AgSt}(H, j, kn_j)$, Pre_j, $Rcv(H, l_3, A_3, m_3) \rightarrow$ $\mathsf{AgSt}(H, j+1, kn_{j+1})$, $Post_{j+1}$, $Snd(H, l_4, A_4, m_4)$
5: **else** ▷ $[\![\Sigma_2]\!]^\mu$ does not contain a transition with $Snd(A_3, l_3, H, m_3)$ in its RHS
6: build all transitions j for all mutations $[\![m_3]\!]^\mu = \{(format(m_3))(m) \mid m \in submsg(m_3)\}$ of m_3 and for each of these, set $[\![kn_{j+1}]\!]^\mu = kn_j \cup \{[\![m_3]\!]^\mu\} \cup Pre_j$ and build all $[\![m_4]\!]^\mu = \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^\mu$, i.e.
7: $[\![kn_{j+1}]\!]^\mu = kn_j \cup \{[\![m_3]\!]^\mu\} \cup Pre_j$,
8: build all $[\![m_4]\!]^\mu = \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^\mu$, i.e.
9: $\mathsf{AgSt}(H, j, kn_j)$, Pre_j, $Rcv(H, l_3, A_3, [\![m_3]\!]^\mu) \rightarrow$ $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu)$, $Post_{j+1}$, $Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
10: **else** ▷ this case is when $kn_j = kn_i \cup X$ for some set X of messages all different from m_3 and received by H in Σ_2 with $X = \{M^x_1, ..., M^x_n\}$
11: **if** $[\![\Sigma_2]\!]^\mu$ still contains a transition with $Snd(A_3, l_3, H, m_3)$ in its RHS **then**
12: H skipped all transitions of Σ_2 in which he received $M^x_1, ..., M^x_n$,
13: $[\![kn_j]\!]^\mu = kn_j = kn_i$,
14: $[\![kn_{j+1}]\!]^\mu = kn_j \cup \{m_3\} \cup Pre_j$,
15: build all $[\![m_4]\!]^\mu = \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[[kn_{j+1}]\!]^\mu$, i.e.
16: $\mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu)$, Pre_j, $Rcv(H, l_3, A_3, m_3) \rightarrow$ $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu)$, $Post_{j+1}$, $Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
17: **else** ▷ $[\![\Sigma_2]\!]^\mu$ does not contain a transition with $Snd(A_3, l_3, H, m_3)$ in its RHS
18: H skipped all transitions of Σ_2 in which he received $M^x_1, ..., M^x_n$ and he cannot receive m_3 in its original form but only in its mutation $[\![m_3]\!]^\mu$
19: $[\![kn_j]\!]^\mu = kn_j = kn_i$,
20: **go to** 6

Algorithm 6 $\mu^H_{skip(RSR)}$: skip $Rcv(H, l_1, A_1, m_1)$ and $Snd(H, A_2, l_2, m_2)$ in i and $Rcv(H, l_3, A_3, m_3)$ in j

1: **if** $[\![kn_j]\!]^\mu = kn_j = kn_i$ **then**
2: $[\![kn_{j+1}]\!]^\mu = kn_j \cup Pre_j$,
3: build all $[\![m_4]\!]^\mu = \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^\mu$, i.e.
4: $\mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu)$, $Pre_j \rightarrow$ $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu)$, $Post_{j+1}$, $Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
5: **else** ▷ this case is when $kn_j = kn_i \cup X$ for some set X of messages all different from m_3 and received by H in Σ_2 with $X = \{m^x_1, ..., m^x_n\}$
6: H skipped all transitions of Σ_2 in which he received $m^x_1, ..., m^x_n$,
7: $[\![kn_j]\!]^\mu = kn_j = kn_i$,
8: **go to** 1

Algorithm 7 Matching mutation for $\mu^H_{skip(R)}$

Consider the transition $next(i)$ that immediately follows the mutated human transition i
$\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
$\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$ where A_s is one of the other agents and l_p and m_p are some channel and message as specified in Σ_2.
if m_2 in $Snd(H, l_2, A_2, m_2)$ in transition i is sent without modification **then**
    $\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
    $\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$
    Let $h ::= next(i)$
    if $\exists\, next(h)$ i.e.
        $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$
    then ▷ no modification are necessary to the next transition
        $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$
        go to 6 with $h ::= next(h)$
else ▷ this case is when $[\![m_2]\!]^\mu$ is sent (modifications have been applied)
    $\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, [\![m_2]\!]^\mu) \rightarrow$
    $\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^\mu),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^\mu)$
    Let $h ::= next(i)$
    if $\exists\, next(h)$ i.e.
        $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$
    then
        $[\![kn_s]\!]^\mu = kn_{s-1}$
        $m_p = [\![m_p]\!]^\mu$
        $[\![kn_{s+1}]\!]^\mu = [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$
        build all $[\![m_{p+1}]\!]^\mu = \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated by $[\![kn_{s+1}]\!]^\mu$ i.e.
        $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},$
        $Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
        go to 13 with $h ::= next(h)$

Algorithm 8 Matching mutation for $\mu^H_{skip(RS)}$

Consider the transition $next(i)$ that immediately follows the mutated human transition i
$\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
$\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$ where A_s is one of the other agents and l_p and m_p are some channel and message as specified in Σ_2.
remove $Rcv(A_2, l_2, H, m_2)$ from $next(i)$
$[\![kn_x]\!]^\mu = kn_{x-1}$
$[\![kn_{x+1}]\!]^\mu = kn_x \cup Pre_x$
build all $[\![m_p]\!]^\mu = \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^\mu$ i.e.
$\mathsf{AgSt}(A_2, x, [\![kn_x]\!]^\mu),\ Pre_x \rightarrow$
$\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^\mu),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^\mu)$
Let $h ::= next(i)$
if $\exists\, next(h)$ i.e.
    $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
    $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$
then
    $[\![kn_s]\!]^\mu = kn_{s-1}$
    $m_p = [\![m_p]\!]^\mu$
    $[\![kn_{s+1}]\!]^\mu = [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$
    build all $[\![m_{p+1}]\!]^\mu = \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated by $[\![kn_{s+1}]\!]^\mu$ i.e.
    $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
    $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},$
    $Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
    go to 9 with $h ::= next(h)$

Algorithm 9 Matching mutation for $\mu^H_{skip(SR)}$

Consider the transition $next(i)$ that immediately follows the mutated human transition i
$\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
$\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$ where A_s is one of the other agents and l_p and m_p are some channel and message as specified in Σ_2.
remove $Rcv(A_2, l_2, H, m_2)$ from $next(i)$
$[\![kn_x]\!]^\mu = kn_{x-1}$
$[\![kn_{x+1}]\!]^\mu = kn_x \cup Pre_x$
build all $[\![m_p]\!]^\mu = \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^\mu$ i.e.
$\mathsf{AgSt}(A_2, x, [\![kn_x]\!]^\mu),\ Pre_x \rightarrow$
$\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^\mu),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^\mu)$
Let $h ::= next(i)$
if $\exists\, next(h)$ i.e.
    $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
    $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$
then
    if s is different than j **then**
        $[\![kn_s]\!]^\mu = kn_{s-1}$
        $m_p = [\![m_p]\!]^\mu$
        $[\![kn_{s+1}]\!]^\mu = [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$
        build all $[\![m_{p+1}]\!]^\mu = \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated by $[\![kn_{s+1}]\!]^\mu$ i.e.
        $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},$
        $Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
        go to 9 with $h ::= next(h)$
    else ▷ s is equal to j, so this is the transition j in which we have to remove the Rcv
        remove $Rcv(A_s, l_p, A_{s-1}, m_p)$ from transition j
        $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y, \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$ is in the form
        $\mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j, \rightarrow$
        $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
        Let $h ::= next(j)$
        if $\exists\, next(h)$ i.e.
            $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
            $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},$
            $Snd(A_s, l_p, A_{s+1}, m_{p+1})$
        then
            $[\![kn_s]\!]^\mu = kn_{s-1}$
            $m_p = [\![m_p]\!]^\mu$
            $[\![kn_{s+1}]\!]^\mu = [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$
            build all $[\![m_{p+1}]\!]^\mu = \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated by $[\![kn_{s+1}]\!]^\mu$ i.e.
            $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
            $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},$
            $Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
            go to 23 with $h ::= next(h)$

Algorithm 10 Matching mutation for $\mu^H_{skip(RSR)}$

Consider the transition $next(i)$ that immediately follows the mutated human transition i
$\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
$\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$ where A_s is one of the other agents and l_p and m_p are some channel and message as specified in Σ_2.
remove $Rcv(A_2, l_2, H, m_2)$ from $next(i)$
$[\![kn_x]\!]^\mu = kn_{x-1}$
$[\![kn_{x+1}]\!]^\mu = kn_x \cup Pre_x$
build all $[\![m_p]\!]^\mu = \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^\mu$ i.e.
$\mathsf{AgSt}(A_2, x, [\![kn_x]\!]^\mu),\ Pre_x \rightarrow$
$\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^\mu),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^\mu)$
Let $h ::= next(i)$
if $\exists\, next(h)$ i.e.
  $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
  $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$
  then
    if s is different than j **then**
      $[\![kn_s]\!]^\mu = kn_{s-1}$
      $m_p = [\![m_p]\!]^\mu$
      $[\![kn_{s+1}]\!]^\mu = [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$
      build all $[\![m_{p+1}]\!]^\mu = \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated by $[\![kn_{s+1}]\!]^\mu$ i.e.
      $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
      $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},$
      $Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
      go to 9 with $h ::= next(h)$
    else ▷ s is equal to j, so this is the transition j in which we have to remove the Rcv
      remove $Rcv(A_s, l_p, A_{s-1}, m_p)$ from transition j
      $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y, \rightarrow$
      $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$ is in the form
      $\mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j, \rightarrow$
      $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
      Let $h ::= next(j)$
      if $\exists\, next(h)$ i.e.
        $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},$
        $Snd(A_s, l_p, A_{s+1}, m_{p+1})$
      then
        $[\![kn_s]\!]^\mu = kn_{s-1}$
        $m_p = [\![m_p]\!]^\mu$
        $[\![kn_{s+1}]\!]^\mu = [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$
        build all $[\![m_{p+1}]\!]^\mu = \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated by $[\![kn_{s+1}]\!]^\mu$ i.e.
        $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},$
        $Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
        go to 24 with $h ::= next(h)$

Algorithm 11 *replace* mutation $\mu^H_{replace}$

build all transitions i obtained by replacing m_2 either with each $[\![m_2]\!]^\mu = \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ or with each $[\![m_2]\!]^\mu$ that is in the powerset of H's current knowledge kn_{i+1} preserving types as specified by the corresponding constants, i.e.
$\mathsf{AgSt}(H, i, kn_i),\ Pre_i, Rcv(H, l_1, A_1, m_1) \rightarrow$
$\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ Snd(H, l_2, A_2, [\![m_2]\!]^\mu)$ ▷ Σ_2 does not contain other transitions by H
if $[\![\Sigma_2]\!]^\mu$ still contains a transition with $Snd(A_3, l_3, H, m_3)$ in its conclusions **then** ▷ the new message $[\![m_2]\!]^\mu$ has no influence on m_3
  $\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Pre_{i+1}, Rcv(H, l_3, A_3, m_3) \rightarrow$
  $\mathsf{AgSt}(H, i+2, kn_{i+2}),\ Post_{i+2},\ Snd(H, l_4, A_4, m_4)$
else
  if $[\![\Sigma_2]\!]^\mu$ contains a transition with $Snd(A_3, l_3, H, [\![m_3]\!]^\mu)$ in its conclusions **then** ▷ the new message $[\![m_2]\!]^\mu$ has some influence on m_3
    $[\![kn_{i+2}]\!]^\mu = kn_{i+1} \cup [\![m_3]\!]^\mu \cup Pre_{i+1}$,
    build all $[\![m_4]\!]^\mu = \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{i+2}]\!]^\mu$, as defined in 1 i.e.
    $\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Pre_{i+1}, Rcv(H, l_3, A_3, [\![m_3]\!]^\mu) \rightarrow$
    $\mathsf{AgSt}(H, i+2, [\![kn_{i+2}]\!]^\mu),\ Post_{i+2},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
  else ▷ $[\![\Sigma_2]\!]^\mu$ does not contain a transition with $Snd(A_3, l_3, H, m_3)$ or $Snd(A_3, l_3, H, [\![m_3]\!]^\mu)$ in its conclusions (the new message $[\![m_2]\!]^\mu$ blocks the sending of m_3)
    $[\![kn_{i+2}]\!]^\mu = kn_{i+1} \cup Pre_{i+1}$,
    build all $[\![m_4]\!]^\mu = \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{i+2}]\!]^\mu$, as defined in 1 i.e.
    $\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Pre_{i+1} \rightarrow$
    $\mathsf{AgSt}(H, i+2, [\![kn_{i+2}]\!]^\mu),\ Post_{i+2},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$

Algorithm 12 Matching mutation for $\mu^H_{replace}$

Consider the transition $next(i)$ that immediately follows the mutated human transition i
$\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
$\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$ where m_2 could be either $[\![m_2]\!]^\mu = \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ or $[\![m_2]\!]^\mu$ that is in the powerset of H's current knowledge kn_{i+1} preserving types as specified by the corresponding constants, A_s is one of the other agents and l_p and m_p are some channel and message as specified in Σ_2.
if $[\![m_2]\!]^\mu = \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ **then**
  $[\![kn_{x+1}]\!]^\mu = kn_x \cup Pre_x \cup [\![m_2]\!]^\mu$
  $[\![m_p]\!]^\mu = m_p$ after removing all the messages that are not in $[\![kn_{x+1}]\!]^\mu$.
else ▷ $[\![m_2]\!]^\mu$ is in the powerset of H's current knowledge kn_{i+1} preserving types as specified by the corresponding constants
  $[\![m_p]\!]^\mu = m_p$ after changing all the messages preserving types as specified by the corresponding constants.
Let $h ::= next(i)$
if $\exists\, next(h)$ i.e.
  $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
  $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$
  then
    $[\![kn_s]\!]^\mu = kn_{s-1}$
    $m_p = [\![m_p]\!]^\mu$
    $[\![kn_{s+1}]\!]^\mu = [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$
    if $[\![m_p]\!]^\mu = \{(format(m_{p-1}))(m) \mid m \in submsg(m_{p-1})\}$ **then**
      $[\![m_{p+1}]\!]^\mu = m_{p+1}$ after removing all the messages that are not in $[\![kn_{s+1}]\!]^\mu$.
    else ▷ $[\![m_p]\!]^\mu$ is generated using H's current knowledge kn_{p-1}
      $[\![m_{p+1}]\!]^\mu = m_{p+1}$ after changing all the messages preserving types as specified by the corresponding constants.
    go to 9 with $h ::= next(h)$

Algorithm 13 *add* mutation μ^H_{add}

Add a transition at state i built by either
  adding a $Snd(H, l, A, m)$ for some l, A and $m \in \mathcal{P}(kn_i) \setminus \emptyset$ preserving types as specified by the corresponding constants, where Pre_i contains only fresh facts (namely those fresh messages needed to built m; hence Pre_i could be empty), keeping the premises fixed as the same as the state i, i.e.
    $\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \rightarrow$
    $\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Snd(H, l, A, m)$
  or duplicating an existing $Snd(H, l, A, m_2)$ action, keeping the premises fixed as the same as the state i, i.e.
    $\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \rightarrow$
    $\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Snd(H, l, A, m_2)$

Algorithm 14 Matching mutation for μ^H_{add}

Consider the new mutated human transition i
    $\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \rightarrow$
    $\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Snd(H, l, A_s, m)$, where A_s is one of the other agents and l and m are some channel and message as specified in Algorithm 13.
Considering the receiver A_s, take its $next(i)$ transition that immediately follows the mutated human transition i.
Create a copy $\overline{next(i)}$ of the $next(i)$ transition and in $\overline{next(i)}$ remove the $Snd()$ event (if any) and replace the values in the $Rcv()$ event with l, m and A_s
    $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_s,\ Rcv(A_s, l, H, m) \rightarrow$
    $\mathsf{AgSt}(A_s, s+1, kn_{s+1})$
For all transitions x after the transition $next(i)$
    $\mathsf{AgSt}(A, x, kn_x) \ldots \rightarrow \mathsf{AgSt}(H, x+1, kn_{x+1}) \ldots$
increment the state increasing the role step the agent is in but keeping the rest of the transitions intact.