A Mutation-Based Approach for the Formal and Automated Analysis of Security Ceremonies

Diego Sempreboni [a] and Luca Viganò [b,*]
[a] *Department of Informatics, King's College London, London, UK*
E-mail: diego.sempreboni@kcl.ac.uk
[b] *Department of Informatics, King's College London, London, UK*
E-mail: luca.vigano@kcl.ac.uk

Abstract. There is an increasing number of cyber-systems (e.g., systems for payment, transportation, voting, critical infrastructures) whose security depends intrinsically on human users. In this paper, we introduce a novel approach for the formal and automated analysis of security ceremonies. A security ceremony expands a security protocol to include human nodes alongside computer nodes, with communication links that comprise user interfaces, human-to-human communication and transfers of physical objects that carry data, and thus a ceremony's security analysis should include, in particular, the mistakes that human users might make when participating actively in the ceremony. Our approach defines mutation rules that model possible behaviors of a human user, automatically generates mutations in the behavior of the other agents of the ceremony to match the human-induced mutations, and automatically propagates these mutations through the whole ceremony. 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 or could be implemented. To automate our approach, we have developed the tool X-Men, which is a prototype that builds on top of 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 three real-life case studies, uncovering a number of concrete vulnerabilities. Some of these vulnerabilities were so far unknown, whereas others had so far been discovered only by empirical observation of the actual ceremony execution or by directly formalizing alternative models of the ceremony by hand, but X-Men instead allowed us to find them automatically.

Keywords: Security ceremonies, Socio-technical security, Formal methods, Mutations, Automated reasoning

1. Introduction

1.1. Context and Motivation

Ellison [1] 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

[*]Corresponding author. E-mail: luca.vigano@kcl.ac.uk.

methodologies or comprehensive toolsets. State-of-the-art approaches and tools for security protocol analysis (e.g., [2–6]) 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 only 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., [7–16]), but they have barely skimmed the surface of taking into account human behavioral and cognitive aspects in their relation with "machine" security.

1.2. 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.[1] More specifically, we provide three main contributions: formalization, tool support and proof-of-concept by means of three case studies.

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. For concreteness, we focus on four main human mutations of a ceremony and their combinations,

- *skipping one or more of the actions that the ceremony expects the human user to carry out* (such as sending or receiving a message),
- *adding an action* (e.g., sending a message twice),
- *replacing a message with another one* (e.g., sending a sub-message of the original message),
- *neglecting to adhere to one or more internal behaviors expected by the ceremony* (such as neglecting to carry out an internal action that is visible only to the agent itself, like a check on the contents of a message),

but our approach is open to extensions with other mutations.

We formalize *algorithms for human mutations*, which take in input a ceremony specification and produce in output a ceremony or a set of ceremonies that result from the mutated actions and/or behavior of the human agent. These algorithms cannot focus only on the mutations of the human, since human ceremony mutations will possibly have an effect also on the other agents of the ceremony, honest or malicious as they may be, humans or processes as they may be. Given a human mutation, we can distinguish two cases:

(1) the other agents are able to reply to the human mutation because the changes are not too relevant or because the ceremony has somehow made provision for it, or

[1]This paper extends and supersedes a preliminary conference version that appeared in [17].

(2) the other agents are not able to reply to the human mutation.

Case (1) may occur, for instance,

- when the ceremony involves the human agent interacting independently with two or more other agents, and the human skips the steps with one agent but still has enough information to exchange messages with the other agents of the ceremony (e.g., when the ceremony is split into independent phases and the human skips phase 1 with agent 1, but is still able to engage in phase 2 with agent 2, without agent 2 noticing that phase 1 did not occur),
- when the human agent adds some message exchanges to a ceremony and the other agents can still participate in the communication (e.g., when the human sends the same message twice and the intended recipient is able to receive both message instances, possibly considering them as messages that are part of two different executions of the same protocol),
- when the human agent replaces a message with another one that can however still be received by the intended recipient (e.g., when the human agents replaces a nonce in the message with a constant that the recipient is however not able to check for freshness),
- when the mutation is only internal to the mutating human agent (e.g., when the agent neglects to check the contents of a message he received), or
- when the other agents are still able to carry out their roles in the ceremony (e.g., when their role includes an if-then-else that captures both original and mutated human behavior; this could happen, for instance, in the implementation of the ceremony, but also in the original ceremony specification where the mutation forces the human to visit by mistake only the else branch of the if-then-else).

In all these subcases, we can simply carry out the analysis of the ceremony after the human mutation.

Human mutations will, however, often yield non-executable models as the agents playing the non-human roles simply won't reply, thus thwarting attacks implicitly or explicitly caused by the human mutation. One might therefore be tempted to discard case (2) as not interesting, but we argue that that is not the case, especially when one considers the way in which the ceremony has actually been implemented or could be implemented.[2] In fact, it is often the case that the implementation of a protocol or ceremony deviates from the original specification. There are several possible reasons for an implementation not to be fully faithful to its specification. 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 and third case studies, the SAML-based Single Sign-on for Google Apps [18] and the Coach Service ceremony). Hence, to investigate whether a human mutation may lead to an attack, in addition to the algorithms for human mutations we formalize *algorithms for matching mutations* for the other agents (where necessary), which allow the other agents to "respond" to the human mutation and continue the execution of the ceremony. The algorithms generate matching mutations from the protocol specification based on the human mutations (the other role specifications are changed to receive/send messages according to the human mutations) and propagate mutations throughout the agents' roles, and thereby create a complete mutated ceremony specification that can be executed and analyzed for attacks. If no attack is found, then we have an additional security guarantee for the ceremony, which is secure even in presence of these mutations. If an

[2]This is interesting not just when the implementation is actually available (in which case one could even try to extract a specification from the code by reverse engineering or model inference), but also as a means to show proactively the pitfalls of a possible implementation.

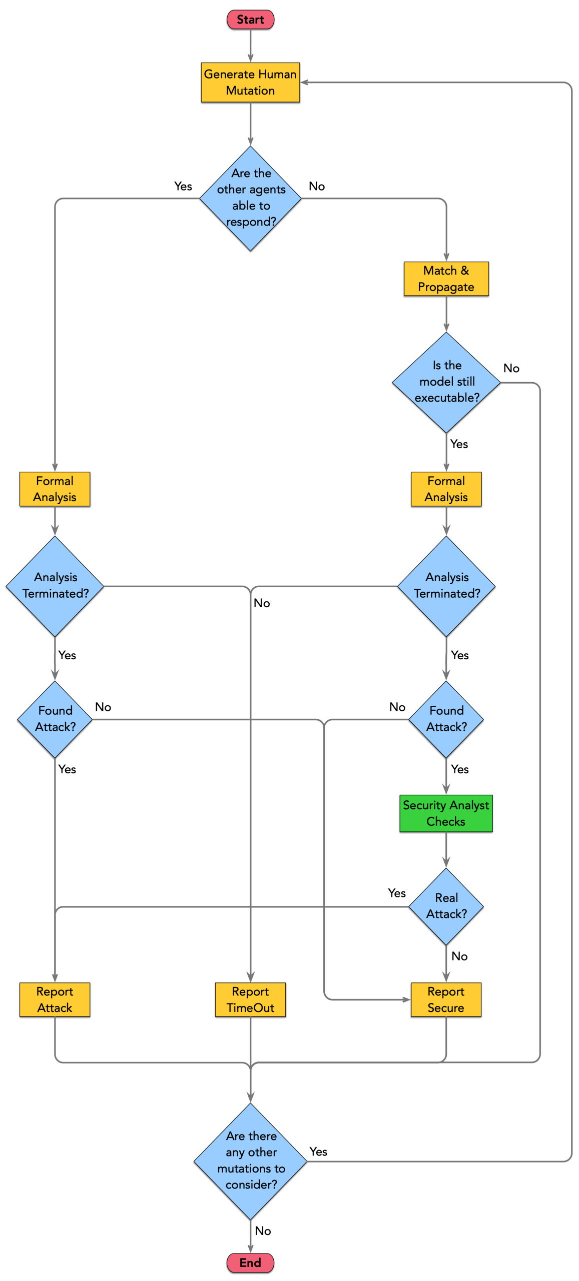


Figure 1. The workflow of our approach

attack is found, then it might be a real attack on the ceremony's (specification and) implementation, or be just a false positive that results from the mutations and is not applicable to the actual implementation. In the spirit of *mutation-based testing* [19–22], the attacks found in this way could be used to generate and apply test cases for the ceremony implementation, but we leave this extension of for future work. Figure 1 provides a summary of the workflow of our approach.

2. *Tool support.* We have developed a prototype tool called *X-Men* (the name was chosen to suggest

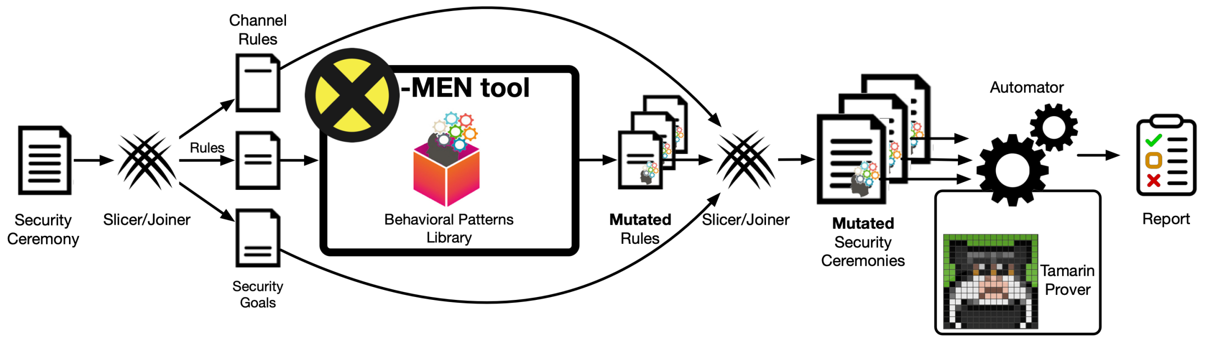


Figure 2. The workflow of the X-Men tool: from models to mutated models that are input to Tamarin

that it considers human mutations), which fully automates the workflow of our approach, except for the green box "Security Analyst Checks" in Figure 1, which is carried out by hand by the analyst. As shown in Figure 2, X-Men creates mutated models that are then automatically used as input to Tamarin [5], 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, e.g., as in the case of the Coach Service ceremony), or it can be used with matching mutations that are propagated to create an executable trace that can be analyzed in search for attacks (as for our two other case studies).

X-Men fully automates the process of generation and analysis of mutated security ceremony models: it takes in input the specification of a ceremony and its goals, along with the choice of mutations to be considered, generates a set of mutated ceremony models, inputs them into Tamarin and then produces a report of the analysis listing all those models for which Tamarin identified vulnerabilities, those for which Tamarin's analysis timed out after 10 minutes, and those that Tamarin was able to verify (i.e., those models for which the analysis terminated without finding any vulnerability). The models for which Tamarin identified vulnerabilities will then need to be inspected by a security analyst to check whether these are vulnerabilities that apply also to the original ceremony and whether the mutations are representative of interesting real-life scenarios. This is similar to what happens with mutation-based testing approaches, where manual intervention of a security analyst is needed (but we point out that tests have shown that we have been able to improve X-Men with respect to the previous version [17] to reduce by almost 50% the number of ceremonies that a security analyst needs to manually investigate). As we remarked above, we leave for future work the extensions that would allow X-Men to generate test cases from the attack traces and to apply them on the ceremony implementation; here we focus on the analysis of models, but we remark that the generation of test cases from attack traces could be done along the lines of [23, 24].

Like for the case in which the automated analysis of a security protocol does not terminate, a security analyst will need to inspect also the models for which Tamarin did not terminate as these might still lead to vulnerabilities (although 10 minutes is often a long enough time for Tamarin to produce an output for short protocols/ceremonies such as the ones we consider in this paper).

3. Proof-of-concept. As a proof-of-concept, we have applied our approach to three real-life case studies, the Oyster Card ceremony, the SAML-based Single Sign-on for Google Apps protocol [18] and a Coach Service ceremony, uncovering a number of concrete vulnerabilities. Some of these vulnerabilities were so far unknown, whereas others had so far been discovered only by empirical observation of the actual ceremony execution or by directly formalizing alternative models of the ceremony by hand, but X-Men instead allowed us to find them automatically.

1.3. Organization

In Section 2, we introduce our motivating and running example, and the other two case studies. In Section 3, we describe the intuitions that underlie our approach. In Section 4, we describe how we formally model security ceremonies and then, in Section 5, we describe how we formally model human mutations of a ceremony. In Section 6, we describe X-Men and its proof-of-concept. In Section 7, we report on the analysis of the three case studies. In Section 8, we discuss related work. In Section 9, we draw conclusions and discuss future work. All our formal models and the code of X-Men are available online at [25]. For readability, we have moved some of the longer figures to the appendix.

2. Three Case Studies

In this section, we describe the three real-life case studies that we consider in this paper, the Oyster Card ceremony, the SAML-based Single Sign-on for Google Apps protocol and a Coach Service ceremony. We begin by introducing the Oyster Card ceremony, which we will use as a motivating and running example.

2.1. The Oyster Card Ceremony

The *Oyster Card* (or just Oyster, for short) is a plastic credit-card-sized, rechargeable, stored-value, contactless smart card used on public transport in Greater London in the United Kingdom. The Oyster is a form of electronic ticket that can hold pay-as-you-go credit, travel cards 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 [26]. Similar systems are in use in a large number of other countries in almost all continents, such as France, Italy, Denmark, Finland, South Africa, Argentina, Chile, Australia and Japan, and, interestingly, most of them suffer from problems similar to the ones of the Oyster that we will discuss in the next sections.

As shown in Figure 3a, 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 chapter, whereas on London buses passengers touch in their Oyster only when boarding (passengers are instead required also to touch out when they alight the bus in Sydney, Australia, and in some other countries like Denmark, for instance). Figure 3b shows an entrance/exit gate of the Tube.

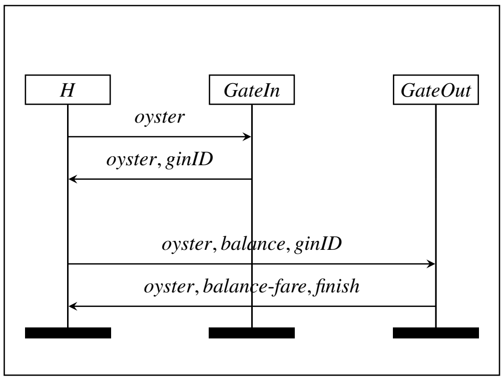
Figure 4a gives a Message Sequence Chart (MSC) of the main Oyster ceremony for the Tube, which is carried out by 3 roles: the human passenger *H*, the entrance gate *GateIn* and the exit gate *GateOut*.

(1) The human passenger *H* touches the 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*, *ginID*, where *ginID* 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 *ginID*.
(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.

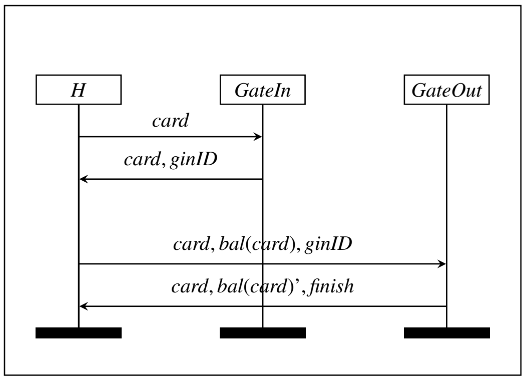
(a) Touching the Oyster on an electronic card reader

(b) A gate of the Tube

Figure 3. Using the Oyster Card in the Tube



(a) The Main Oyster ceremony for the Tube



(b) The Generalized Main ceremony for the Tube

Figure 4. The ceremonies for the Tube

Some remarks are in order. First of all, note that we did not obtain this specification from TfL, 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 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 models, and in fact the language that we describe below does contain cryptographic operators.[3]

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.

[3]Note also that initial versions of the MiFare chip, and thus of the Oyster, suffered from a number of attacks [27–29], but the current version of the Oyster does not suffer from these problems any more since it is based on the new MiFare DESFire family that uses stronger encryption algorithms.

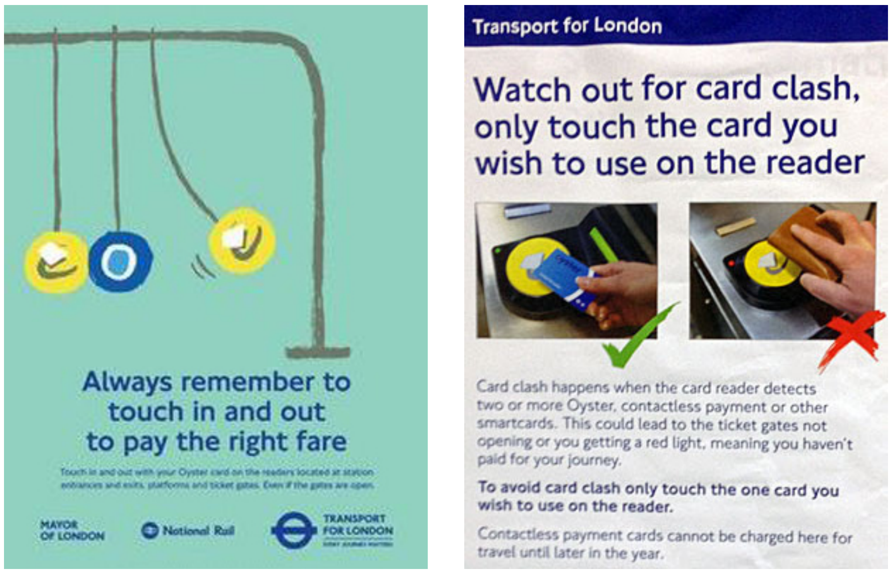

Always remember to touch in and out to pay the right fare
Touch in and out with your Oyster card on the readers located at station entrances and exits, platforms and ticket gates. Even if the gates are open.
MAYOR OF LONDON
National Rail
TRANSPORT FOR LONDON
Transport for London
Watch out for card clash, only touch the card you wish to use on the reader
Card clash happens when the card reader detects two or more Oyster, contactless payment or other smartcards. This could lead to the ticket gates not opening or you getting a red light, meaning you haven't paid for your journey.
To avoid card clash only touch the one card you wish to use on the reader.
Contactless payment cards cannot be charged here for travel until later in the year.

Figure 5. Warnings issued to the Tube passengers

Fourth, the ceremony in Figure 4a 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 4a 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 4b. 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 3a, 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 5. The poster on the left of Figure 5 reminds Tube 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 [30]. 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 5 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* [31]. 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.

Before we explain in detail how we formally model and reason about security ceremonies, let us first introduce our two other case studies.

2.2. The SAML-based Single Sign-on for Google Apps Protocol

Single Sign-on (SSO) protocols enable entities (companies, associations, institutions or universities, etc.) to establish a federated environment in which clients sign in the environment once and are then able to access services offered by different providers in the federation. The *Security Assertion Markup Language 2.0 Web Browser SSO Profile* is one of the standards on which several established software companies have based their SSO implementations. Google, for instance, developed a SAML-based SSO for its Google Applications Premier Edition, a service for using custom domain names with several Google web applications such as Gmail, Google Calendar and Google Docs.

The profile is shown in Figure 6, which, like the other figures for this case study, we have taken directly from [18]. The profile works as follows: a client C aims at getting access to a service or a resource that is located at the address URI and is provided by the service provider SP, which is one of the service providers in the federated environment; to that end, SP issues an authentication request of the form $AuthReq(ID, SP)$, where ID is a string uniquely identifying the request; the identity provider IdP, who presides over the federated environment as the certification authority able to authenticate the different clients, challenges C to provide valid credentials (this is typically taken care of at the beginning of the run when the client types in its username and password) and, upon successful authentication, IdP builds an authentication assertion $AuthAssert(ID, C, IdP, SP)$ and digitally signs it with its private key K_{IdP}^{-1}, which is denoted symbolically by writing $\{AuthAssert(ID, C, IdP, SP)\}_{K_{IdP}^{-1}}$; SP checks the authentication assertion and, acknowledging the power of the IdP to issue authentication assertions in the federated environment, provides the resource to C.

Carrying out a digital signature of a long message such as the authentication assertion is computationally expensive and might slow down the interaction between the user and the machine, so, in their implementation of this protocol in 2008, Google's engineers simplified the authentication assertion by removing two of its parameters, ID and SP, as shown in Figure 7. They simplified $AuthAssert(ID, C, IdP, SP)$ into $AuthAssert(C, IdP)$ to speed up the digital signature, but it turned out that this allowed a malicious SP to use this assertion to pose as C in another run of the protocol.

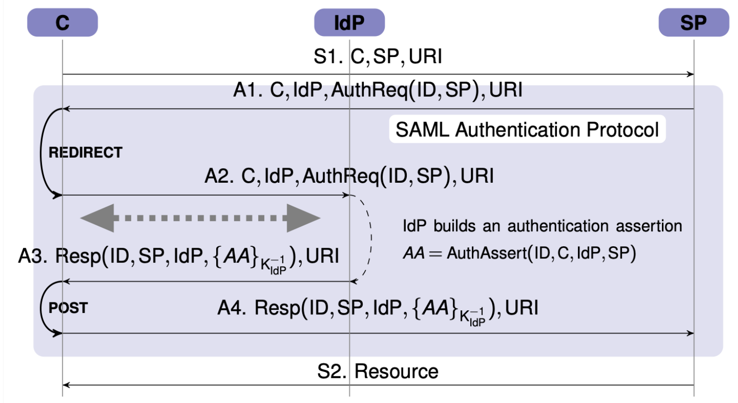

C
IdP
SP
S1. C, SP, URI
A1. C, IdP, AuthReq(ID, SP), URI
SAML Authentication Protocol
REDIRECT
A2. C, IdP, AuthReq(ID, SP), URI
IdP builds an authentication assertion
AA = AuthAssert(ID, C, IdP, SP)
A3. Resp(ID, SP, IdP, {AA}K−1IdP), URI
POST
A4. Resp(ID, SP, IdP, {AA}K−1IdP), URI
S2. Resource

Figure 6. SAML 2.0 Web Browser SSO Profile (from [18])

Armando et al. [18] formalized a model of this protocol and, by using SATMC, a model checker for security protocols that is embedded in both the AVANTSSAR Platform [2] and the SPaCIoS Tool [24], they were able to discover the attack that is illustrated in Figure 8 by considering the following scenario. Let *IdP* be a hospital, *C* be a doctor employed by the hospital and *SP* a medical insurance company. The doctor *C* initiates the protocol in order to get access to a resource provided by the insurance company *SP*, say to update the information about one of the doctor's patients who is insured by *SP*. However, as usual, there is no guarantee that the agents participating in a protocol are all honest and a dishonest insurance company (as denoted by the orange bandit) might want to acquire confidential information about the patient. Thanks to the simplified signed authentication assertion $\{AuthAssert(C, IdP)\}_{K_{IdP}^{-1}}$, which in this scenario is $\{AuthAssert(doc, H)\}_{K_{H}^{-1}}$, the dishonest insurance company is able to start another run of the protocol in which it engages with another service provider but pretending to be the doctor. For instance, the other service provider could be Google Mail or Docs and the insurance company could log in as the doctor and read confidential information about the patient in the doctor's email or online documents. The dishonest service provider thus acts as a man-in-the-middle that participates in two runs of the protocols: one in which it acts under its real name (as the service provider *SP*) and one in which it pretends to be the client (the doctor *doc*).

The attack is due to *IdP* simply (and wrongly) issuing a simplified certificate $\{AuthAssert(C, IdP)\}_{K_{IdP}^{-1}}$ that intuitively says "I certify that this is the client *C*. Signed: The Identity Provider *IdP*", so that when the dishonest service provider is shown the certificate, it can first copy it and then claim to be the client *C* in another protocol run. The solution to the man-in-the-middle attack is to restore the deleted information in the authentication assertion, as Google did in the new implementation of their protocol that they produced after Armando et al. notified them about the attack.

Armando et al. were able to discover the attack by writing a formal model of the protocol as it had been implemented by Google, with the wrongly simplified authentication assertion, and then analyzing the model using SATMC. In this paper, instead, we show how the simplified and vulnerable version of the authentication assertion can be obtained by a human *IdP* producing a mutated assertion, and how X-Men allows us to re-discover the attack automatically.

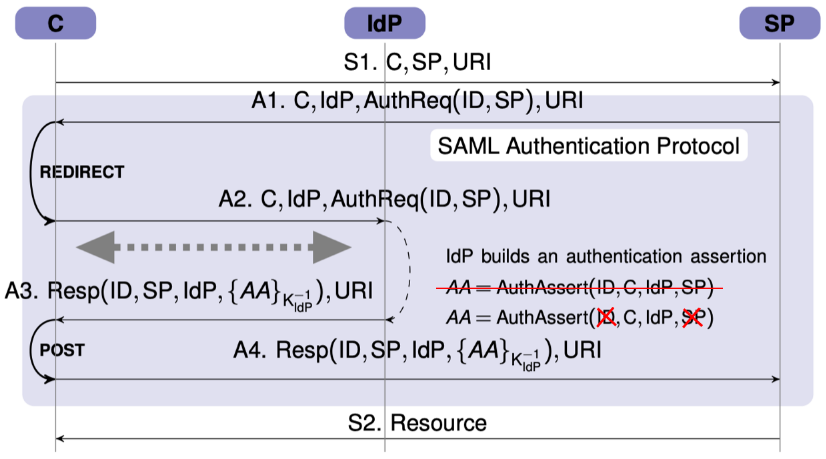

C
IdP
SP
S1. C, SP, URI
A1. C, IdP, AuthReq(ID, SP), URI
SAML Authentication Protocol
REDIRECT
A2. C, IdP, AuthReq(ID, SP), URI
IdP builds an authentication assertion
~~AA = AuthAssert(ID, C, IdP, SP)~~
AA = AuthAssert(ID, C, IdP, SP)
A3. Resp(ID, SP, IdP, {AA}K⁻¹IdP), URI
POST
A4. Resp(ID, SP, IdP, {AA}K⁻¹IdP), URI
S2. Resource

Figure 7. The SAML-based Single Sign-on for Google Apps protocol (from [18])

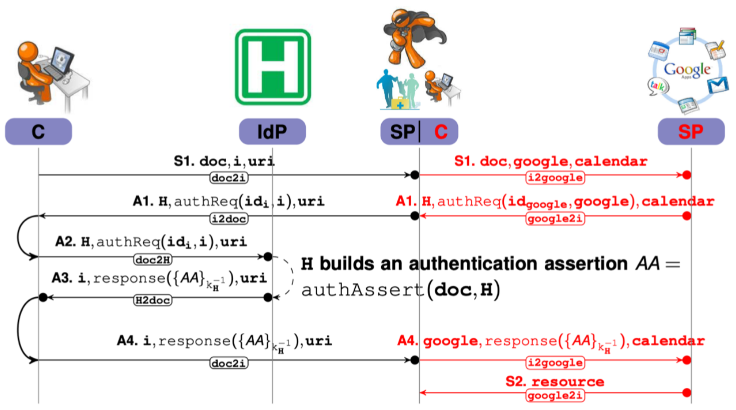

C
IdP
SP | C
SP
S1. doc, i, uri
doc2i
S1. doc, google, calendar
i2google
A1. H, authReq(idi, i), uri
i2doc
A1. H, authReq(idgoogle, google), calendar
google2i
A2. H, authReq(idi, i), uri
doc2H
H builds an authentication assertion AA = authAssert(doc, H)
A3. i, response({AA}kH⁻¹), uri
H2doc
A4. i, response({AA}kH⁻¹), uri
doc2i
A4. google, response({AA}kH⁻¹), calendar
i2google
S2. resource
google2i

Figure 8. The man-in-the-middle attack on the SAML-based SSO for Google Applications (from [18])

2.3. A Coach Service Ceremony

People get to and from airports thanks to different transportation services such as underground, trains, taxis or coaches (i.e., buses or similar vehicles). As our third case study, we consider a security ceremony for (the inspection of tickets in) coach services.

The majority of coach services offer essentially two types of tickets: *paper tickets* and *electronic tickets*, or *e-tickets* for short (we do not consider smart-card tickets because they would require a different kind of attack and, anyway, smart-card solutions are not as widely adopted for coach services, contrarily to urban buses). Customers can buy fixed-price paper tickets physically in shops at the airports, in coach stations or directly from the coach driver on the day of travel. Alternatively, they can buy e-tickets online on the website (or the app) of the coach service. The procedure to buy an e-ticket requires the customer to choose the journey specifying the date of travel (or dates in case of a return ticket, unless it is an "open" return ticket, which has a flexible validity), insert the customer's details and pay for the ticket.

Once the payment is concluded, the customer receives a confirmation email for the payment, which also acts as the e-ticket for the journey.

Different coach services offer e-tickets that share several similarities in terms of information present on the ticket itself. We considered these similarities to define a realistic standalone e-ticket based on the information that many different tickets of many different coach services have in common. Our standalone e-ticket has the following fields: *customer name*, *ticket number*, *departure date*, *return date* (if it is a return ticket), *departure time*, expected *arrival time*, *from* (which specifies where the journey starts), *to* (which specifies where the journey ends) and the *price* of the ticket. In case the e-ticket is a return ticket, there are additionally fields for the return departure time and return arrival time.

A number of coach services (as well as other services that offer e-tickets) enrich the e-ticket with one or more visual fields like a barcode or a QR code: these aim to simplify and speed up the ticket inspection process. To that end, the driver is equipped with a device capable of recognizing the information within the codes so that such information does not have to be controlled manually. The e-ticket may also contain other information and the terms of condition, which we have omitted here as they are not relevant for the description nor for the attack that we will discuss later.

There is no available generic and public specification of the ceremony for coach services and how tickets are bought, delivered and inspected. Thus, rather than considering the specific ceremony of a specific service, we have observed many different such services in different cities and then synthesized them into a single plausible Coach Service ceremony.[4] In a nutshell, a customer purchases a ticket and shows it to the driver, who checks the ticket and if it deems it to be valid, allows the customer to board the coach. There are two ticket inspections sub-ceremonies: one for paper tickets and one for e-tickets.

The sub-ceremony for paper tickets is shown in Figure 9a. The driver receives the paper ticket from the customer and then checks the journey information such as from/to and the departure time. If this information is correct, then the driver admits the customer on the coach. Note that there is no real check on the genuineness of the ticket.

The sub-ceremony for e-tickets keeps the same spirit of the sub-ceremony for paper tickets, but delegating some of the checks to technology (or so one would expect). Here too, upon request of the driver, the customer shows the e-ticket that comes as the confirmation email of the payment for the journey. The driver performs a preliminary analysis of the e-ticket to understand the type of the e-ticket checking also the presence of possible visual fields, such as the check of a QR-code as is shown in the example of sub-ceremony in Figure 9b. Based on the presence of visual fields, the driver can make two different choices in order to validate a ticket:

- check if unique details printed on the ticket (e.g., ticket number) are present on a list of allowed tickets for that journey;
- use the visual fields reading them with an electronic reader.

If the driver decides to check the unique details in the ticket manually, he performs a visual inspection, looking for a match of any of the unique details (e.g., ticket number, journey references, service numbers, travel route references, etc.) with a list in his possession (e.g., electronic list using a pad, a printed list). Once he finds a match, the driver validates the ticket and admits the customer. For instance, the driver can first identify the ticket number on the e-ticket, and then check whether the ticket number matches a

[4]So, both our ceremony and tickets are realistic rather than real, but this is more than enough for our purposes in this paper. We have reviewed a handful of coach services in the UK and abroad, but note that we cannot provide more details on our observations as that might allow readers to identify the specific coach services or, even worse, identify specific drivers and ticket inspectors.

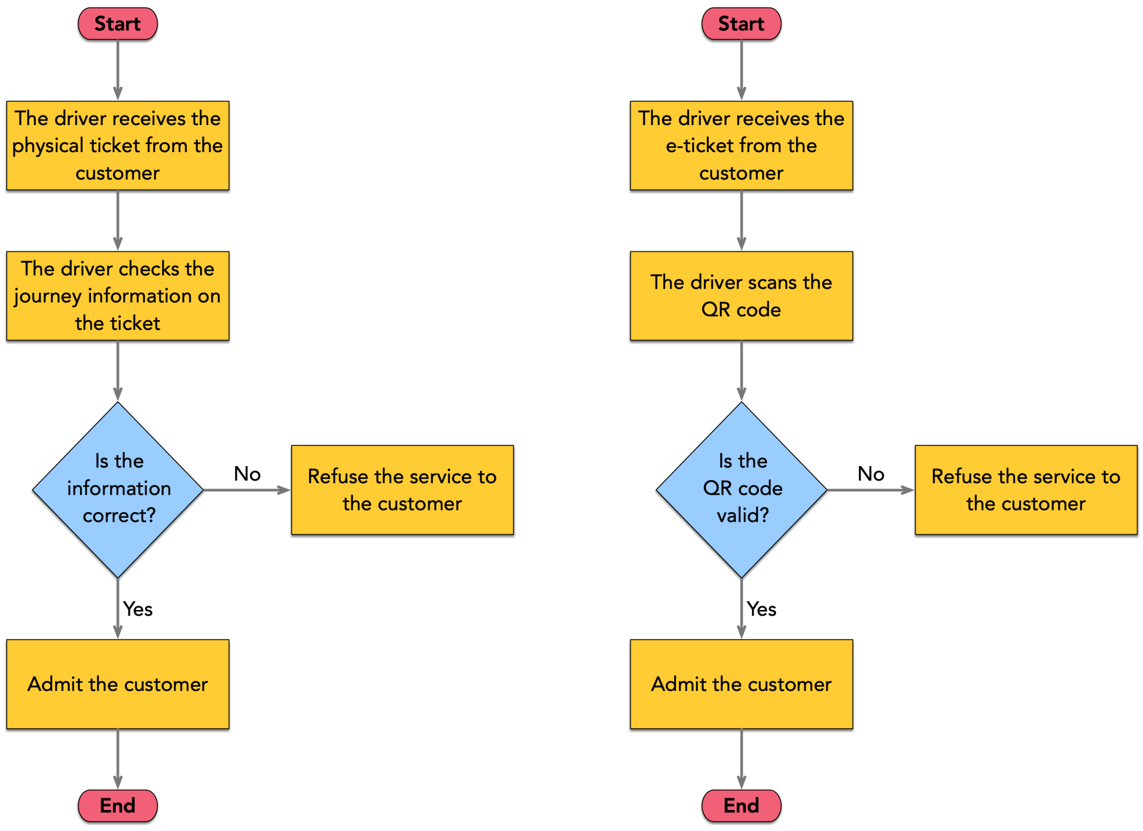

Start
The driver receives the physical ticket from the customer
The driver checks the journey information on the ticket
Is the information correct?
No
Refuse the service to the customer
Yes
Admit the customer
End
Start
The driver receives the e-ticket from the customer
The driver scans the QR code
Is the QR code valid?
No
Refuse the service to the customer
Yes
Admit the customer
End
(a) The Paper Ticket Inspection sub-ceremony (b) An example of E-Ticket Inspection sub-ceremony

Figure 9. The Ticket Inspection Sub-ceremonies

list of allowed ticket numbers for that journey on an electronic pad (that synchronizes regularly with the main system); once (and if) a match is found, the driver validates the ticket number to make it unavailable for the future and admits the customer.

If the driver decides to use the visual fields, he needs to scan them (or a selection, e.g., QR code, barcode, etc.) with a device: once the device confirms that the ticket is valid, the driver admits the customer.

For concreteness, we can then consider the Coach Service ceremony for e-tickets that is shown in Figure 10. The ceremony has three roles: the *Customer*, the *Driver* and the online website *WebServer* of the coach service.

(1) The *Customer* interrogates the *WebServer* in order to get a list of coaches operating on a specific date *date* at a specific time *dtime*, departing from a specific location *from* and arriving at a specific destination *to*.
(2) The *WebServer* returns the available solution(s), indicating the price *price*.
(3) The *Customer* clicks on the selected option, paying the price of the ticket.[5]

[5]The actual online purchase process might, of course, suffer from several possible vulnerabilities, including some that are potentially due to human mistakes, but we do not consider them in this paper.

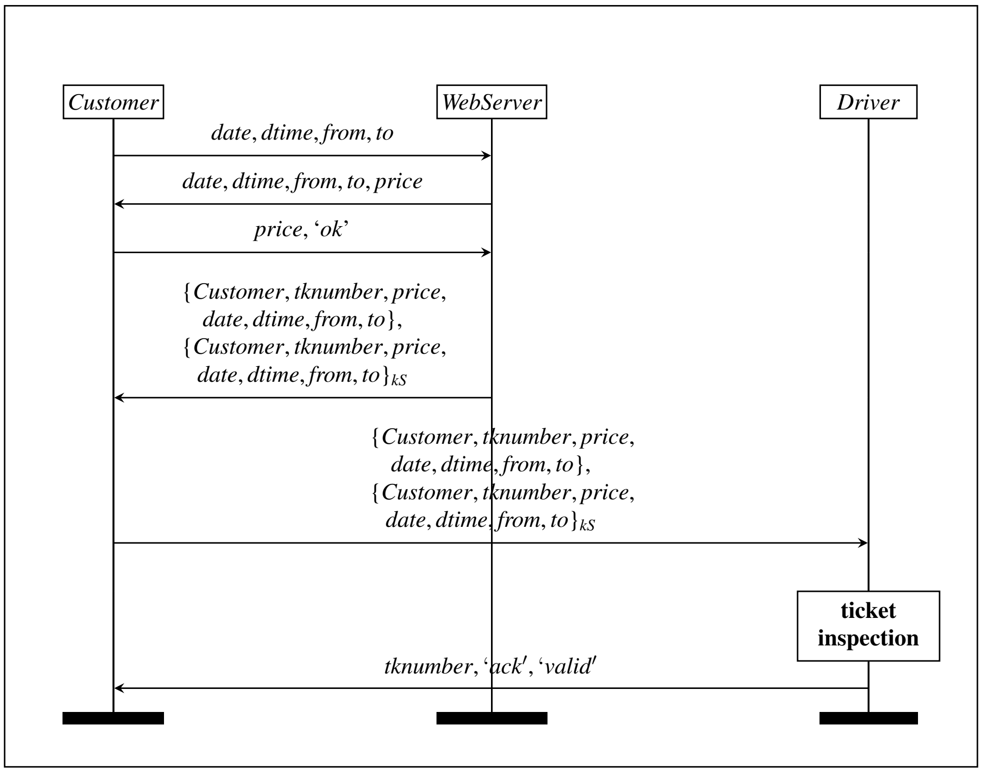

Customer
WebServer
Driver
date, dtime, from, to
date, dtime, from, to, price
price, 'ok'
{Customer, tknumber, price, date, dtime, from, to}, {Customer, tknumber, price, date, dtime, from, to}kS
{Customer, tknumber, price, date, dtime, from, to}, {Customer, tknumber, price, date, dtime, from, to}kS
ticket inspection
tknumber, 'ack', 'valid'

Figure 10. The Coach Service ceremony (for e-tickets)

(4) The *WebServer* sends to the *Customer* the e-ticket with the related information, encrypting them using a shared key *kS* between the *WebServer* and the *Driver*. Encrypting the fields represents the fact that the *Driver* recognizes the validity of the e-ticket; this could be seen as applying a watermarking on the e-ticket.
(5) The *Customer*, at the time of travel, shows the e-ticket to the *Driver*.
(6) The *Driver* performs a ticket inspection and, if successful, admits the *Customer* to the coach.

A preliminary security analysis of the tickets and of the ceremony is beneficial to understand possible vulnerabilities. In particular, we looked at flaws that could allow an attacker to travel for free using forged tickets bypassing the ceremony's security measures.

Typically, coach services can rely on the difficulty of physically forging accurate copies of paper tickets, which would require specific equipment such as the proper paper, a printing machine, the identical layout the coach service is using on a digital file, etc., which are not easy to obtain. These difficulties are reflected in the lower number of checks that the Paper Ticket Inspection sub-ceremony puts in place with respect to the e-ticket one. Figure 9a shows that, typically, no specific visual checks are performed, in particular, no checks of unique fields (e.g., ticket number). The checks are limited to the plausibility of the information only.

E-tickets, on the other hand, represent a different scenario. Coach services rely (or should rely) on the difficulty of obtaining unique information displayed on the e-ticket to discourage forging. This unique information can also be used by the coach service companies to generate visual fields that can be printed on the tickets. However, during our investigation of the processes followed by different services, we encountered a number of visual fields (e.g., a machine-readable representation of numerals and characters such as a barcode or a QR code) that, even though they could have potentially stored relevant information for the ceremony, did not contain any information but were just used as graphical means to corroborate the structure of the e-tickets. This is a potential attack vector: a coach service that offers customers to choose between different types of e-tickets such as QR code or barcode with no information (e.g., in case of specific ticket such as discount tickets) can be exploited to execute a successful forging attack.

In the case of e-tickets, the precision with which the checks are carried out is crucial. The inspector (e.g., the driver) has to perform proper checks on the unique fields displayed on the ticket. However, as we observed, drivers often apply the Paper Ticket Inspection sub-ceremony also for e-tickets, even though the Paper Ticket sub-ceremony is not designed to be secure against forged e-tickets. As mentioned above, the Paper Ticket sub-ceremony does not consider checks on visual fields and relies on the complicated procedure to forge paper tickets.

Forging e-tickets, on the other hand, is considerably easier and does not require much expertise or technical knowledge. One can use a raster graphics editor such as Adobe Photoshop (or even simpler editors like GIMP) or one can directly use an email client. In fact, after having purchased an e-ticket, a confirmation email is sent to the customer's email address. This email also works as ticket, which can also be saved/exported as PDF file (as most of the email clients allow one to do). Once in possession of the email, an attacker can forge an e-ticket by modifying it after having exported or received it in PDF format. Alternatively, a less elaborated method to forge e-tickets is for the attacker to modify the confirmation email using the email client and to forward the new email to the attacker's own address.

If the driver, by mistake, applies the Paper Ticket Inspection sub-ceremony to an e-ticket accurately forged, then the driver will accept the forged e-ticket and the attacker will be able to use the service for free. To validate these insights, we formalized a mutation that captures the behavioral pattern that gives rise to mistakes similar to the one made by the driver (namely, to neglect carrying our proper checks on the data received) and we have applied it to analyze the Coach Service ceremony in Figure 10.

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* [32] or the vertical lines in MSCs and *Alice&Bob notation* [33], or less graphically by a process in a process algebra such as in the *Applied Pi Calculus* [34]. 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 11. When complete, the jigsaw puzzle produces a complete picture: a trace of the ceremony.

Now, we have all been there: you are trying to assemble one of those really difficult jigsaw puzzles, 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

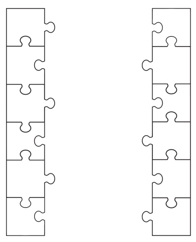


Figure 11. A simple ceremony between a User (left) and a System (right) depicted as a jigsaw puzzle

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 12: if the human user does not know how long the edge should be, then he could terminate it by attaching the piece pictured in red as in Figure 12b; or the human could add one more piece to the edge as in Figure 12c; or the human could append a wrong piece pictured in red as in Figure 12d, which raises the question of how the remaining pieces would fit (they are thus drawn with dotted lines).

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

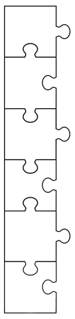
- *skip* some actions, e.g., touching out with an Oyster without having touched in with any card, as illustrated in Figure 12b by the anticipated termination of the role;
- *add* some actions, e.g., touch in with two cards, as illustrated in Figure 12c by the additional piece;
- *replace* an action with another one, e.g., using a contactless credit card to touch out instead of the Oyster he used to touch in, as illustrated in Figure 12d by the different outgoing connector, which represents a different message being sent;
- *neglect* to carry out one or more role-actions, e.g., neglecting to carry out a check on the contents of a message (such as forgetting to check if the balance is positive), as illustrated in Figure 12e.[6]

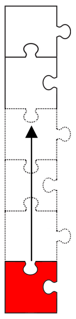
In our approach, we represent these human "mistakes" as *mutations* with respect to the role as specified originally — hence the name "X-Men" for our tool, which captures the fact that we are considering mutations of the original human behavior. Such a mutation does not just have a local effect (for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

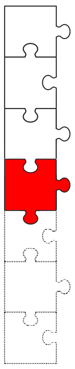
It is, however, not enough to simply allow the human to carry out these unforeseen actions (add, skip, replace or neglect some parts of the role). In order to reason about what would happen if the human carried out these mutations, we need to capture the fact that a mutation of the human behavior will likely have an effect also on the other agents of the ceremony. More specifically, note the difference between the role-action in the neglect mutation, which is an "internal" action as it only involves the mutating agent, and the actions in the skip, add and replace mutations, which are instead actions that involve not only the mutating agent, but also, albeit indirectly, other agents in the ceremony.

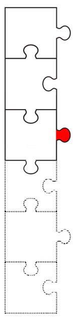
To illustrate this, consider again, for simplicity, the ceremony between User and System in Figure 11 and consider the scenario in which a human playing the role User replaces an event of his role with a

[6]Role-actions, which we will define at the beginning of Section 4.2, are internal actions carried out by the roles in the ceremony, such as sending or receiving messages, generating fresh messages or checking the contents of messages received.


(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 (skip mutation)


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


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

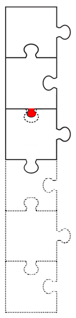

(e) The role User as carried out by a human who connects a piece that is identical to the expected one except for some of the checks on the incoming message (neglect mutation)

Figure 12. A human carrying out the role User... and mutating it, by mistake or lack of understanding

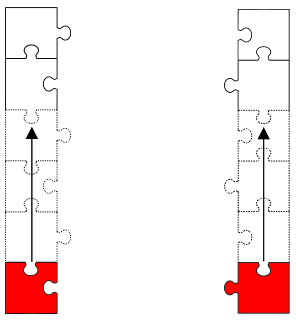


Figure 13. The skip mutation of the role User (as in Figure 12b) and the matching mutation of the role System

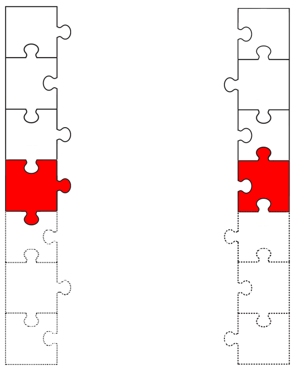


Figure 14. The add mutation of the role User (as in Figure 12c) and the matching mutation of the role System

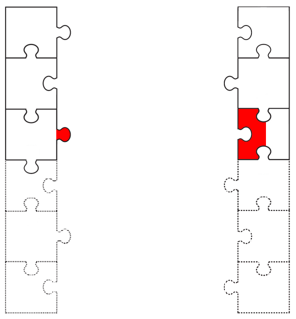


Figure 15. The replace mutation of the role User (as in Figure 12d) and the matching mutation of the role System

different one, i.e., sends a message m' instead of the specified message m, as depicted in Figure 12d. As we discussed in the introduction, there are two cases. Let us elaborate more on what we already remarked by using the Oyster ceremony as a concrete example.

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.[7] 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. Could it, however, potentially lead to an attack? For instance, what about the ceremony's implementation? Does the current implementation really conform to the specification (and will a future implementation conform)? 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. However, 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, or neglect a check) 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'".[8] To reason about such a situation, we pair the mutation of the User role with a *matching mutation* of the System role to receive/send messages according to the human mutation, and thereby generate an executable trace of the ceremony. For instance, in the case of the Oyster example, if the human User sends the same message or a similar message twice, then we change the System to receive both messages, whereas if the User skips a step, then we change the System to skip a step correspondingly and continue the ceremony (we will return to this in Section 5.1.1, in which we will give a concrete example of a mutation and its matching mutation).

This is in line with *mutation-based testing* [19–22], 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 and that 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 11, our approach mutates the role of the System as shown in Figure 13 to match the human User's skip mutation of Figure 12b. 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 (the mutation is propagated), which we again illustrate with dotted lines.

Similarly, for the same ceremony between User and System in Figure 11, our approach mutates the role of the System as shown in Figure 14 to match the human User's add mutation of Figure 12c, and it mutates the role of the System as shown in Figure 15 to match the human User's replace mutation of Figure 12d.

[7]It could even happen because also the role System is played a human who does a mistake, as we discuss in the case of the neglect mutation.

[8]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 5 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.

Finally, it is interesting to note that there is no matching mutation for the human User's neglect mutation of Figure 12e as that mutation is local to the human and has no effect on the other roles of the ceremony.

If these matching mutations lead to an attack, then we can check (possibly discussing with the ceremony designers) whether the mutated specification makes sense and, in any case, use the obtained attack trace to generate concrete test 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.

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 human actions or neglect role-actions) and the matching mutations of the other roles of the ceremony. The resulting mutated ceremony models 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 [23, 24]).

4. Formal Modeling of Security Ceremonies

We import, adapt and extend notions that are used in many of the state-of-the-art approaches and tools for the formal analysis of security protocols. For concreteness, our tool X-Men builds on top of the Tamarin prover [5, 7, 8] to model and analyze security ceremonies with mutations caused by human users, but our approach is for the most part general and independent of Tamarin and could be applied similarly to other tools such as ProVerif [3], Maude-NPA [4], the AVANTSSAR Platform [2] and its follow-up the SPaCIoS Tool [24].

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

Definition 1 (Algebra of messages)**.** *The* term algebra of messages *is given by $\mathcal{T}_\Sigma(\mathcal{V})$, where Σ is a signature and $\mathcal{V}$ is a countably infinite set of variables. Σ contains three sets $\mathcal{C}_{fresh}$, $\mathcal{C}_{pub}$ and $\mathcal{F}$ that are pairwise disjoint: $\mathcal{C}_{fresh}$ is a countably infinite set of* fresh constants *modeling the generation of nonces, $\mathcal{C}_{pub}$ is a countably infinite set of* public constants *representing agent names and other publicly known values, and $\mathcal{F}$ is a finite set of* function symbols *that includes symbols for*

- *the* pairing $pair(m_1, m_2)$ *of two message terms m_1 and m_2,*
- *the* first projection $\pi_1(m_1)$ *and* second projection $\pi_2(m_1)$ *of a pair m_1 of message terms,*
- *the* hash $h(m_1)$ *of a message term m_1,*
- *the* symmetric encryption $senc(m_1, m_2)$ *of a message term m_1 with a message term m_2,*
- *the* symmetric decryption $sdec(m_1, m_2)$ *of a message term m_1 with a message term m_2,*
- *the* asymmetric encryption $aenc(m_1, m_2)$ *of a message term m_1 with a message term m_2,*
- *the* asymmetric decryption $adec(m_1, m_2)$ *of a message term m_1 with a message term m_2,*
- *the* signature $sign(m_1, m_2)$ *of a message term m_1 with a message term m_2, and the corresponding* verification $verify(m_1, m_2, m_3)$ *of a signature message term m_1 with message terms m_2 and m_3.*

The function $pk(m_1)$ represents the public key corresponding to the private key m_1. For readability, we will also use the variables k to denote keys and, for brevity, we will denote $pair(m_1, m_2)$ also by

$\langle m_1, m_2 \rangle$ *and we will write, say,* $\langle m_1, m_2, m_3 \rangle$ *for* $\langle m_1, \langle m_2, m_3 \rangle \rangle$ *when there is no risk of confusion for the projections. We might also just write, e.g.,* m_1, m_2 *for* $\langle m_1, m_2 \rangle$.

A message term *m is* ground *when it contains no variables. We will also call a message term simply* message *or* term.

Having defined the algebra of messages, we can now define what is a submessage of a message and what is the format of message.

Definition 2 (Submessage and format of a message)**.** *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* $m_1 \in submsg(m_3)$;
- $m_2 = \circ(m_3, k)$ *for some* m_3 *and* k*, with* $\circ \in \{senc, aenc, sign\}$*, 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* $f = m$;
- *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* $f = \circ$.

We adopt the equational theory for messages proposed by Dolev and Yao [35], which defines equations on the relationships between operators for composing and decomposing messages.

Definition 3 (Dolev-Yao-style equational theory)**.** *Messages are composed and decomposed using the standard* Dolev-Yao-style equational theory *for the functions in* $\mathcal{F}$*, based on the equations*

- $\pi_1(\langle m_1, m_2 \rangle) = m_1$ *and* $\pi_2(\langle m_1, m_2 \rangle) = m_2$,
- $sdec(senc(m_1, m_2), m_2) = m_1$,
- $adec(aenc(m_1, pk(m_2)), m_2) = m_1$,
- $verify(sign(m_1, m_2), m_1, pk(m_2)) = true$.

However, as we do for instance for the Oyster ceremony in Section 7.1, our approach allows us also not to consider explicitly the presence of a (Dolev-Yao) attacker, and instead focus on capturing the way human agents might interact insecurely with the other ceremony agents even in the absence of an attacker. 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 and the Coach Service ceremony (see Section 7.2 and Section 7.3, respectively), we add an explicit attacker who intentionally tries to make the ceremony insecure.[9] In all these cases, our modeling of the human behavior (through mutations of 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.

[9]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.

4.2. Ceremony Specification

To formally specify and analyze security ceremonies, we adopt standard notions used for security protocol analysis in Tamarin, e.g., [5, 7, 8]. We will try to be thorough but also succinct and readers interested to know more about Tamarin are advised to refer to the papers, to Tamarin's user manual [36] and its website http://tamarin-prover.github.io. Note that Tamarin grants quite a lot of freedom to modelers, and one might, for instance, find keywords with the same name but different arity in different papers. For this reason, in the following we clarify the notions and symbols that we use in this paper.

Formally, a *role script* is a sequence of events $e \in \mathcal{T}_{\Sigma \cup RoleActions}(\mathcal{V})$, where $RoleActions = \{Snd, Rcv, Start, Fresh\}$ is a set of action names with their respective arity as defined below, and each event e has exactly one function symbol that is in *RoleActions* at its top-level (i.e., as its format). We will introduce other Tamarin actions in our specifications. To avoid confusion, in the following we will explicitly speak of "human actions" (in a ceremony) and of "role-actions" (in a role specification). As notation, we denote *sequences* with square brackets; for instance, in Section 5, 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.

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, $l \in LinkProp = \{ins, auth, conf, sec\}$ indicates the type of channel over which a message is sent, $P \in \mathcal{C}_{pub}$ is a role's name, and $m \in \mathcal{T}_{\Sigma}(\mathcal{V})$ is a message. The channel types *ins*, *auth*, *conf* and *sec* denote insecure, authentic, confidential, and secure channels, respectively, and correspond in the obvious manner to the channel symbols in the Alice&Bob notation (see [8] as well as [33, 37, 38] for a detailed discussion of different types of channels, including pseudonymous channels).

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.

The *fresh* event $Fresh(A, m)$ indicates that the role A generates a fresh message m (e.g., a nonce or a new key).

The *start* event is the first event of a role script and occurs only once: $Start(A, K)$ indicates the *initial knowledge* K of agent A, which is the set of ground terms that A knows at the beginning of the ceremony, such as the A's own public and private keys, public keys of other agents, symmetric keys shared with other agents, etc. The *knowledge of agent A* starts with A's initial knowledge and increases monotonically during the execution of the ceremony as A receives messages or generates fresh terms.

As a concrete example, let us return to the Generalized Main ceremony for the Tube. As shown in Figure 4b, this ceremony 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 and, in fact, we do not consider an explicit attacker (but we easily could, and indeed we do so for the two other example ceremonies we consider, cf. Sections 4.3.1 and 4.3.2). We represent this by specifying that all messages are sent over secure channels. Thus, the role scripts for the roles of this ceremony are the ones given in Figure 16.

We take advantage of *constants* in Tamarin to identify values received and sent during a ceremony. In [8], 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*, denoted in Tamarin's notation by writing the name of the type in single quotes, e.g., '*type*'. In this paper, we only consider types of ground terms, such as the type '*card*' for *card* or '*balance*' for $bal(oyster)$ as shown in the role scripts and in the agent rules in Figure 17, Figure 18 and Figure 19.[10]

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

$$
\begin{aligned}
&\mathit{RoleScript}_H = \\
&\quad [\mathit{Start}(H, \langle\langle \text{`}\mathit{GateIn}\text{'}, \text{`}\mathit{GateOut}\text{'}, \text{`}\mathit{card}\text{' `}\mathit{balance}\text{'}\rangle, \langle \mathit{GateIn}, \mathit{GateOut}, \mathit{card}, \mathit{bal}(\mathit{card})\rangle\rangle), \\
&\quad\ \mathit{Snd}(H, \mathrm{sec}, \mathit{GateIn}, \langle \text{`}\mathit{card}\text{'}, \mathit{card}\rangle), \\
&\quad\ \mathit{Rcv}(H, \mathrm{sec}, \mathit{GateIn}, \langle\langle \text{`}\mathit{card}\text{'}, \text{`}\mathit{ginID}\text{'}\rangle, \langle \mathit{card}, \mathit{ginID}\rangle\rangle), \\
&\quad\ \mathit{Snd}(H, \mathrm{sec}, \mathit{GateOut}, \langle\langle \text{`}\mathit{card}\text{'}, \text{`}\mathit{balance}\text{'}, \text{`}\mathit{ginID}\text{'}\rangle, \langle \mathit{card}, \mathit{bal}(\mathit{card}), \mathit{ginID}\rangle\rangle), \\
&\quad\ \mathit{Rcv}(H, \mathrm{sec}, \mathit{GateOut}, \langle\langle \text{`}\mathit{card}\text{'}, \text{`}\mathit{balance}\text{'}, \text{`}\mathit{finish}\text{'}\rangle, \langle \mathit{card}, \mathit{bal}(\mathit{card})', \mathit{finish}\rangle\rangle)] \\
&\mathit{RoleScript}_{\mathit{GateIn}} = \\
&\quad [\mathit{Start}(\mathit{GateIn}, \langle H, \mathit{ginID}\rangle), \\
&\quad\ \mathit{Rcv}(\mathit{GateIn}, \mathrm{sec}, H, \langle \text{`}\mathit{card}\text{'}, \mathit{card}\rangle), \\
&\quad\ \mathit{Snd}(\mathit{GateIn}, \mathrm{sec}, H, \langle\langle \text{`}\mathit{card}\text{'}, \text{`}\mathit{ginID}\text{'}\rangle, \langle \mathit{card}, \mathit{ginID}\rangle\rangle)] \\
&\mathit{RoleScript}_{\mathit{GateOut}} = \\
&\quad [\mathit{Start}(\mathit{GateOut}, \langle H, \mathit{gout}\rangle), \\
&\quad\ \mathit{Rcv}(\mathit{GateOut}, \mathrm{sec}, H, \langle\langle \text{`}\mathit{card}\text{'}, \text{`}\mathit{balance}\text{'}, \text{`}\mathit{ginID}\text{'}\rangle, \langle \mathit{card}, \mathit{bal}(\mathit{card}), \mathit{ginID}\rangle\rangle), \\
&\quad\ \mathit{Snd}(\mathit{GateOut}, \mathrm{sec}, H, \langle\langle \text{`}\mathit{card}\text{'}, \text{`}\mathit{balance}\text{'}, \text{`}\mathit{finish}\text{'}\rangle, \langle \mathit{card}, \mathit{bal}(\mathit{card})', \mathit{finish}\rangle\rangle)]
\end{aligned}
$$

Figure 16. The role scripts for the roles of the Generalized Main ceremony for the Tube

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 on we will often omit constants in role scripts and rules, so that when we write m, the reader should please mentally replace it with the constant-message pair* $\langle t, m\rangle$.

4.3. Execution Model

Our approach is based on Tamarin’s *execution model* [5], which is defined by a *multiset term-rewriting system* like in many other security protocol analysis tools (e.g., [2, 4, 38]). In the following, we recall from the Tamarin manual and papers the definitions of the notions of states, facts, traces, protocol specification and its rules. A state is a multiset of facts that model resources, including the information that agents know and exchange. More formally:

Definition 4 (States and facts)**.** *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 is a finite sequence of multisets of role-actions that is generated by multiset rewriting.

Definition 5 (Traces)**.** *A* trace *tr is a finite sequence of multisets of role-actions a and is generated by the application of labeled* state transition rules *of the form*

$$\mathit{prem} \xrightarrow{a} \mathit{conc}\,.$$

$$[] \xrightarrow{Start(H,\langle\langle\text{'}GateIn\text{'},\text{'}GateOut\text{'},\text{'}card\text{'}\ \text{'}balance\text{'}\rangle,\langle GateIn,GateOut,oyster,balance\rangle\rangle)} [\mathsf{AgSt}(H,1,\langle GateIn, GateOut, oyster, balance\rangle)] \qquad (H_0)$$

$$\begin{aligned}&[\mathsf{AgSt}(H,1,\langle GateIn, GateOut, oyster, balance\rangle)] \\ &\xrightarrow{Snd(H,\text{sec},GateIn,\langle\text{'}card\text{'},oyster\rangle)} \\ &\quad [\mathsf{AgSt}(H,2,\langle GateIn, GateOut, oyster, balance\rangle), \\ &\quad \mathsf{Out}_{sec}(H, GateIn, \langle\text{'}card\text{'}, oyster\rangle)]\end{aligned} \qquad (H_1)$$

$$\begin{aligned}&[\mathsf{AgSt}(H,2,\langle GateIn, GateOut, oyster, balance\rangle),\ \mathsf{In}_{sec}(GateIn, H, \langle\langle\text{'}card\text{'},\text{'}ginID\text{'}\rangle,\langle oyster, ginID\rangle\rangle)] \\ &\xrightarrow{Rcv(H,\text{sec},GateIn,\langle\langle\text{'}card\text{'},\text{'}ginID\text{'}\rangle,\langle oyster,ginID\rangle\rangle)} \\ &\quad [\mathsf{AgSt}(H,3,\langle GateIn, GateOut, oyster, balance, ginID\rangle)]\end{aligned} \qquad (H_2)$$

$$\begin{aligned}&[\mathsf{AgSt}(H,3,\langle GateIn, GateOut, oyster, balance, ginID\rangle)] \\ &\xrightarrow{Snd(H,\text{sec},GateOut,\langle\langle\text{'}card\text{'},\text{'}balance\text{'},\text{'}ginID\text{'}\rangle,\langle oyster,bal(oyster),ginID\rangle\rangle)} \\ &\quad [\mathsf{AgSt}(H,4,\langle GateIn, GateOut, oyster, balance, ginID\rangle), \\ &\quad \mathsf{Out}_{sec}(H, GateOut, \langle\langle\text{'}card\text{'},\text{'}balance\text{'},\text{'}ginID\text{'}\rangle, \langle oyster, bal(oyster), ginID\rangle\rangle)]\end{aligned} \qquad (H_3)$$

$$\begin{aligned}&[\mathsf{AgSt}(H,4,\langle GateIn, GateOut, oyster, balance, ginID\rangle), \\ &\mathsf{In}_{sec}(GateOut, H, \langle\langle\text{'}card\text{'},\text{'}balance\text{'},\text{'}finish\text{'}\rangle, \langle oyster, bal(oyster)', finish\rangle\rangle)] \\ &\xrightarrow{Rcv(H,\text{sec},GateOut,\langle\langle\text{'}card\text{'},\text{'}balance\text{'},\text{'}finish\text{'}\rangle,\langle oyster,bal(oyster)',finish\rangle\rangle),EndJourney(H,\text{'}card\text{'},oyster)} []\end{aligned} \qquad (H_4)$$

Figure 17. The rules for the human agent in the Generalized Main ceremony for the Tube

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 role-actions in a in the trace. The set of all traces of a set of rules $\mathcal{R}$ *is denoted by TR*$(\mathcal{R})$.

Consider, for instance, a *generic* ceremony between a human agent H and possibly pairwise distinct agents $A_1, A_2, A_3, A_4, \ldots$. Trace (1) denotes a trace of the role-actions of the agents in this ceremony,

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

$$\begin{aligned}&[\mathsf{AgSt}(GateIn, 1, \langle H, ginID\rangle), \mathsf{In}_{sec}(H, GateIn, \langle\text{`}card\text{'}, oyster\rangle)]\\ &\quad\xrightarrow{Rcv(GateIn,\mathrm{sec},H,\langle\text{`}card\text{'},oyster\rangle)}\\ &\qquad[\mathsf{AgSt}(GateIn, 2, \langle H, ginID, oyster\rangle]\end{aligned} \tag{Gi_1}$$

$$\begin{aligned}&[\mathsf{AgSt}(GateIn, 2, \langle H, ginID, oyster\rangle]\\ &\quad\xrightarrow{Snd(GateIn,\mathrm{sec},H,\langle\langle\text{`}card\text{'},\text{`}ginID\text{'}\rangle\,\langle oyster,ginID\rangle\rangle),CommitIdentifier(GateIn,H,ginID)}\\ &\qquad[\mathsf{Out}_{sec}(GateIn, H, \langle\langle\text{`}card\text{'}, \text{`}ginID\text{'}\rangle, \langle oyster, ginID\rangle\rangle)]\end{aligned} \tag{Gi_2}$$

Figure 18. Agent rules for the *GateIn* in the Generalized Main ceremony for the Tube

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

$$\begin{aligned}&[\mathsf{AgSt}(GateOut, 1, \langle H, gout\rangle),\ \mathsf{In}_{sec}(H, GateOut, \langle\langle\text{`}card\text{'}, \text{`}balance\text{'}, \text{`}ginID\text{'}\rangle, \langle oyster, bal(oyster), ginID\rangle\rangle)]\\ &\quad\xrightarrow{Rcv(GateOut,\mathrm{sec},H,\langle\langle\text{`}card\text{'},\text{`}balance\text{'},\text{`}ginID\text{'}\rangle,\langle oyster,bal(oyster),ginID\rangle\rangle)}\\ &\qquad[\mathsf{AgSt}(GateOut, 2, \langle H, GateOut, oyster, bal(oyster), ginID\rangle]\end{aligned} \tag{Go_1}$$

$$\begin{aligned}&[\mathsf{AgSt}(GateOut, 2, \langle H, GateOut, oyster, bal(oyster), ginID\rangle]\\ &\quad\xrightarrow{Snd(GateOut,\mathrm{sec},H,\langle\langle\text{`}card\text{'},\text{`}balance\text{'},\text{`}finish\text{'}\rangle,\langle oyster,bal(oyster)',\text{`}finish\text{'}\rangle\rangle),TouchOut(GateOut,H,\text{`}finish\text{'})}\\ &\qquad[\mathsf{Out}_{sec}(GateOut, H, \langle\langle\text{`}card\text{'}, \text{`}balance\text{'}, \text{`}finish\text{'}\rangle, \langle oyster, bal(oyster)', \text{`}finish\text{'}\rangle\rangle)]\end{aligned} \tag{Go_2}$$

Figure 19. Agent rules for the *GateOut* in the Generalized Main ceremony for the Tube

where each Σ_i represents a possibly empty subtrace:

$$\begin{array}{l}\qquad\vdots\Sigma_1\\ Rcv(H, l_1, A_1, m_1)\\ Snd(H, l_2, A_2, m_2)\\ Rcv(A_2, l_2, H, m_2)\\ \qquad\vdots\Sigma_2\\ Rcv(H, l_3, A_3, m_3)\\ Snd(H, l_4, A_4, m_4)\\ Rcv(A_4, l_4, H, m_4)\\ \qquad\vdots\Sigma_3\end{array} \tag{1}$$

When $A_1 = A_2 = \ldots = A$, trace (1) reduces to a trace of a ping-pong ceremony between H and a system A.

Definition 6 (Protocol model)**.** *A* protocol model *consists of the agent rules, the fresh rule, channel rules and attacker rules.*

These rules are defined in the remainder of this section. 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 [7]) to model the sending and receiving of messages over authentic/confidential/secure channels, and thus control the ability of the attacker (who, e.g., cannot 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 17, whereas the rules for the *GateIn* and *GateOut* agents are in Figure 18 and Figure 19, respectively (and the rules for the SSO and Coach Service ceremonies are in Section 4.3.1 and Section 4.3.2, respectively). 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 $\mathsf{AgSt}(A, step, kn_{step})$, and *conc* contains the subsequent agent state fact $\mathsf{AgSt}(A, step+1, kn_{step+1})$, where *step* refers to the role step the agent is in and kn_{step} 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 $\mathsf{AgSt}(A, 0, m)$.[11]

We have given the rules of the Oyster case study. Before we introduce the goals of the case studies and how they can be formalized in Tamarin's language, let us give the rules for the two other ceremonies we consider in this paper.

4.3.1. Rules of the SSO ceremony

The agent rules for the *Client*, *IdP* and *SP* in the SSO ceremony are in Appendix A.1 (in Figure 25, Figure 26 and Figure 27, respectively). These rules model the ceremony as it is depicted in Figure 6, where a client interacts with an identity provider and a service provider to be able to use its services.

4.3.2. Rules of the Coach Service ceremony

The agent rules for the *Customer*, *WebServer* and *Driver* in the Coach Service are in Appendix A.2 (in Figure 28, Figure 29 and Figure 30, respectively). These rules model the ceremony as it is depicted in Figure 10, where a customer of a coach service buys an e-ticket from the webserver of the coach

[11]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 $l \in \{ins, conf\}$.

service company and then uses it to travel after having interacted with a driver. Note that in the rule (D_2)

$$
\begin{array}{l}
[\mathsf{AgSt}(\mathit{Driver}, 2, \langle \mathit{kS}, \mathit{Customer}, \mathit{tknumber}, \mathit{date}, \mathit{dtime}, \mathit{from}, \mathit{to}\rangle)] \\
\qquad \xrightarrow{\substack{\mathit{Snd}(\mathit{Driver},\mathrm{sec},\mathit{Customer},\langle \mathit{tknumber},\mathit{date},\text{‘}\mathit{ack}\text{’},\text{‘}\mathit{valid}\text{’}\rangle) \\ \mathit{Eq}(\mathit{tknumber},\mathit{tknumber}_{kS}), \\ \mathit{Eq}(\mathit{Customer},\mathit{Customer}_{kS}), \\ \mathit{Eq}(\mathit{price},\mathit{price}_{kS}), \\ \mathit{Eq}(\mathit{date},\mathit{date}_{kS}), \\ \mathit{Eq}(\mathit{dtime},\mathit{dtime}_{kS}), \\ \mathit{Eq}(\mathit{from},\mathit{from}_{kS}), \\ \mathit{Eq}(\mathit{to},\mathit{to}_{kS})}} \\
\qquad\quad [\mathsf{AgSt}(\mathit{Driver}, 3, \langle \mathit{kS}, \mathit{Customer}, \mathit{tknumber}, \mathit{date}, \mathit{dtime}, \mathit{from}, \mathit{to}\rangle), \\
\qquad\quad \mathsf{Out}_{sec}(\mathit{Driver}, \mathit{Customer}, \langle \mathit{Customer}, \mathit{tknumber}, \mathit{date}, \text{‘}\mathit{ack}\text{’}, \text{‘}\mathit{valid}\text{’}\rangle)] \qquad (D_2)
\end{array}
$$

Tamarin actions are used to model the check the *Driver* carries out during the ticket inspection. This check represents a verification of the genuineness of the data encrypted (e.g., using the function *Eq* to check that the fields *Customer*, *tknumber*, *price*, *date*, *dtime*, *from*, *to* match their respective encrypted fields of the encrypted e-ticket) during the phase of releasing the ticket by the *WebServer*. Practically, it could consist in checking a watermark on a ticket, or leaving the check to a device that is connected with the *WebServer* itself. To avoid misconceptions, signed fields of the tickets are renamed by affixing *kS* at the end of the fields.

4.4. 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 Section 2.1, “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 run. We formalize this in our Tamarin specifications by including explicit restrictions (through the *OnlyOnce* restriction [36] in the Setup phase; see our models in [25]) to force the human to carry out a single journey in the ceremony. While these three goals capture the main warnings issued by TfL (cf. Section 2.1), one could of course consider a number of other goals, e.g., variants of *GO3* that formalize the case in which the human touches in with two cards (regardless of whether he then touches out with one or two of these cards) or touches out with two cards (regardless of whether he touched in with one or two of these cards), or goals that span multiple journeys (i.e., multiple ceremony runs). We leave the investigation of such goals for future work as our aim here is not to carry out an exhaustive analysis of the Oyster ceremony or of the other case studies, but rather to use the goals we consider as examples for our proof-of-concept on how one can use our approach for the analysis of security ceremonies.

We can formalize these three goals using Tamarin's notation as shown in Section 4.4.1, whereas in Section 4.4.2 and Section 4.4.3 we formalize some of the goals of the SSO ceremony and of the Coach Service ceremony, respectively.

4.4.1. Goal for the Oyster ceremony

Goal *GO1* is formalized in Tamarin's language by the lemma in Listing 1, which, in addition to `Rcv`, uses the following role-actions, which already appeared in the agent rules, to express that certain actions have occurred during the ceremony:

- `EndJourney(H,'card',oyster)` expresses that the agent `H` ends the journey, where a journey is considered ended when the human has passed through a gate out after touching the `oyster` card,
- `CommitIdentifier(GateIn,H,ginID)` expresses that the agent `GateIn` writes the identifier `ginID` of the gate on the card possessed by the agent `H` (note that we do not need to specify the card in this role-action, but rather the gate "gives" the identifier to the human).

```
lemma complete journey: all-traces
"All H oyster #j.
EndJourney(H,'card',oyster)@j
==> (Ex GateIn l ginID #i. Rcv(GateIn,l,H,oyster)@i
& CommitIdentifier(GateIn,H,ginID)@i
& i<j)"
```

Listing 1: Lemma for the security goal *GO1*

The security goal *GO1* can be read as follows: if an agent `H` ends a journey with an Oyster card `oyster` at time instant `j` (as specified by `EndJourney(H,'card',oyster)@j`), which also represents the end of the ceremony, then there is a previous time instant `i`, where a `GateIn` has received from `H` the same Oyster card `oyster` used to end the journey (as specified by `Rcv(GateIn,l,H,oyster)@i`) and `GateIn` wrote its identifier `ginID` on the card (as specified by `CommitIdentifier(GateIn,H,ginID)@i`).

Goal *GO2* is formalized in Tamarin's language by the lemma in Listing 2, which can be read as follows: if an agent `H` ends a journey with an Oyster card `oyster` at time instant `j` (as specified by `EndJourney(H,'card',oyster)@j`), which also represents the end of the ceremony, then there is a previous time instant `t` in which `H` sent, touching the card on the reader, the card information to the gate in (as specified by `Snd(H,l,'card',oyster)@t`), and there does not exist a time instant `c`, which can be equal or not to `t`, in which another card `ccard`, different from `oyster`, is touched on the reader of the gate in (as specified by `Snd(H,l,'card',ccard)@c`). In short, the agent did not touch in another card.

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

Listing 2: Lemma for the security goal *GO2*

It is important to note here the connection between the role-action `CommitIdentifier` used in *GO1* and the role-action `Snd` used in *GO2*. Both role-actions try to capture the action in a moment where the card touches on the reader of the `GateIn`. However, while `CommitIdentifier(GateIn,H,ginID)` models the point of view of `GateIn` with the writing of its identifier (denoted also by the fact that `GateIn` is the first argument and that `CommitIdentifier` is only used within `GateIn`'s agent rules), `Snd(H,l,'card',oyster)` models the point of view of the agent `H` sending the information of the Oyster card `oyster` (`H` is the first argument and this `Snd` is only used within `H`'s agent rules).

Goal *GO3* is formalized in Tamarin's language by the lemma in Listing 3, which uses the role-action `TouchOut(GateOut,H,'finish')` to express that the human agent has touched a card at a gate out; when an agent touches a card at the gate out, the gate out signals the end of the journey by means of `'finish'`.

```
lemma Card_Clash_Out: all-traces
"All H GateIn GateOut oyster ginID l1 l2 #j #t.
Rcv(GateOut,l1,H,oyster)@j
& TouchOut(GateOut,H,'finish')@j
& Rcv(GateIn,l2,H,oyster)@t
& CommitIdentifier(GateIn,H,ginID)@t
& t<j
& not (GateIn = GateOut)
==> not (Ex ccard l #i #k.
Rcv(GateOut,H,ccard)@i
& TouchOut(GateOut,H,'finish')@i
& Rcv(GateIn,l,H,ccard)@k
& CommitIdentifier(GateIn,H,ginID)@k
& k<i
& not (oyster = ccard))"
```

Listing 3: Lemma for the security goal *GO3*

The security goal *GO3* can be read as follows: if an agent `GateOut` receives an Oyster card `oyster` from agent `H` at time instant `j` (as specified by `Rcv(GateOut,l,H,oyster)@j`) and signals the end of the journey at that time instant (as specified by `TouchOut(GateOut,H,'finish')@j`), and a `GateIn`, different from `GateOut`, receives the Oyster card `oyster` from `H` and writes its identifier `ginID` to `H` at a previous time instant `t` (as specified by `CommitIdentifier(GateIn,H,ginID)@t`), then there does not exist a card `ccard` different from `oyster` such that `GateOut` and `GateIn` execute the same transitions with that `ccard` at time instants `i` and `k`, respectively.

It is important to note here the connection between the role-action `EndJourney` used in both *GO1* and *GO2* and the role-action `TouchOut` used in *GO3*. Both role-actions try to capture the action of a card that has been touched out on the reader of the gate out. However, while `TouchOut(GateOut,H,'finish')` models `GateOut`'s point of view, (denoted also by the fact that `GateOut` is the first argument and that `TouchOut` is only used within `GateOut`'s agent rules), `EndJourney(H,'card',oyster)` models `H`'s point of view (`H` is the first argument and `EndJourney` is only used within `H`'s agent rules).

4.4.2. Goal for the SSO ceremony

For the SSO ceremony, we have formalized the goal "*IdP* authenticates only the agent who requires to be authenticated" as a standard injective-agreement goal in Tamarin as in Listing 4. Other goals could of course be considered for a full-fledged analysis of the ceremony, but again here we are mainly focused on showing our approach at work rather than carrying out such a full-fledged analysis. This goal is formalized in Tamarin's language by the lemma in Listing 4, which uses, in particular, the following role-actions:

- `Commit(actor, peer, params)` expresses that some parameters `params` are committed by an `actor` to a `peer`,
- `Running(actor, peer, params)` expresses that an `actor` is participating in the ceremony with a `peer` using some parameters `params`,
- `RevLtk(actor)` expresses that a long-term private key belonging to the agent `actor` has been revealed, meaning that the agent is compromised.

```
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)"
```

Listing 4: Lemma for the security goal of SSO

This security goal can be read as follows: if some parameters `params` are committed by an `actor` to a `peer` at time instant `i` (as specified by `Commit(actor, peer, params)@i`), then there exists a time instant `j` at which `actor` is participating in the ceremony with the `peer` using the same parameters (as specified by `Running(actor, peer, params)@j`) and there does not exist another actor `actor2` who has committed the same parameters used by `actor` to another peer `peer2` at time instant `i2` different from `i` (as specified by `Commit(actor2, peer2, params)@i2`). Otherwise, either the long term keys of actor or the long term keys of peer have been revealed to allow another `actor2` and `peer2` to commit the same `params`.

4.4.3. Goal for the Coach Service ceremony

We consider one security property: *authenticity of the ticket*. Other properties (e.g., ensuring that customers are billed properly or the other security properties for the Oyster ceremony) can be modeled and analyzed similarly. This goal is formalized in Tamarin's language by the lemma in Listing 5, which uses, in particular, the following role-actions:

- `End(Customer,'ack','valid',tknumber,date)` expresses that the agent `Customer` receives the acknowledgment `ack` that the presented ticket, with number `tknumber` and date `date` is `valid`.
- `ValidTicket(WebServer,Customer,'tknumber',tknumber,date)` expresses that a ticket, with number `tknumber`, has been issued by the `WebServer` for the `Customer` for the date `date`.

```
lemma auth: all-traces
"All Customer tknumber date #i.
End(Customer,'ack','valid',tknumber,date)@i
==> Ex WebServer #j.
ValidTicket(WebServer,Customer,'tknumber',tknumber,date)@j
& j<i"
```

Listing 5: Lemma for the authenticity of the ticket in the Coach Service ceremony

The Coach Service ceremony verifies the lemma `auth` if, when a `Customer` completes the ceremony at time instant `i` with a ticket `tknumber` valid for a specific `date` (as specified by `End(Customer,'ack','valid',tknumber,date)@i`, then there is a previous time instant `j` at which the ticket `tknumber` was issued by `WebServer` for `Customer` for the date `date` (as specified by `ValidTicket(WebServer,Customer,'tknumber',tknumber,date)@j`).

As for the Oyster ceremony, this goal refers to a single ticket inspection ceremony, i.e., a single ceremony run, that comprises also the purchase phase in which that specific ticket has been bought. If an agent is able to end the ceremony with a ticket that has not been bought for that specific journey (e.g., the agent has changed the date of an old legit ticket to make the ticket valid once again), then the security goal is falsified as the two date fields in the role-actions `End` and `ValidTicket` do not match. More complex scenarios could be considered for a full-fledged analysis of the ceremony.

5. 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. As we remarked above (cf., in particular, Figure 1), 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 mirror the skip of the human actions. By introducing the human mutations, matching them appropriately (if and when needed), and propagating these mutations (if and when needed) throughout the specification, we will eventually obtain fully mutated ceremony traces, which our tool feeds into Tamarin, first to check if they are executable and then to analyze them with respect to the corresponding goal(s). Of course, in some cases, depending on what is mutated, a human mutation might yield a full ceremony trace without having to match this mutation and propagate the changes.

We first introduce, using a fairly high-level language, the notion of a generic human mutation along with the notion of a matching mutation.

Definition 7 (Generic human mutation and matching mutation)**.** *A generic* human mutation *is a function*

$$\mu^H : tr \mapsto tr'$$

that takes as input a trace tr and gives as output a new trace $\mu^H(tr) = tr' = [\![tr]\!]^{\mu^H}$ obtained by mutating H's subtrace as a consequence of the human H "deviating" from the original role script by skipping one or more human actions, replacing a message with another one, adding a new human action, or neglecting one or more role-actions.

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

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

In the following subsections, we will instantiate these generic definitions to define the four human mutations *skip*, *add*, *replace* and *neglect* both formally and algorithmically, giving also the algorithmic definitions of the corresponding matching mutations (Definitions 8, 9, 10 and 11, respectively).

Slightly abusing notation, we will write $[a_0, \dots, a_i, \dots, a_n]^H$ with $0 < i \leqslant n$ to denote the subtrace of a human agent H in a ceremony execution, and we will apply $[\![\_]\!]^\mu$ not just to traces, but also to actions, messages and knowledge. 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 preconditions and postconditions. More specifically, in the style of multiset rewriting as in [39], we consider an abstract "merged" transition rule in which *prem* contains the receipt of a message and *conc* the sending of the reply:[12]

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

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 and the

[12]This is in the spirit of the *step compression* technique that is adopted in several security protocol analysis tools, such as [2]. The idea is that some actions can be safely lumped together. For instance, we can safely assume that if a role is supposed to send a reply to a message it received, then we can compress the receive and send actions into a single transition.

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

Then, the trace (1) can be rewritten as follows, where we now embed A_2's receipt of m_2 in Σ_2 and A_4's receipt of m_4 in Σ_3 as we wish to focus on H's actions:

$$
\begin{array}{c}
\vdots \Sigma_1 \\
\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \to \\
\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ Snd(H, l_2, A_2, m_2) \\
\vdots \Sigma_2 \\
\mathsf{AgSt}(H, j, kn_j),\ Pre_j,\ Rcv(H, l_3, A_3, m_3) \to \\
\mathsf{AgSt}(H, j+1, kn_{j+1}),\ Post_{j+1},\ Snd(H, l_4, A_4, m_4) \\
\vdots \Sigma_3
\end{array}
\tag{2}
$$

5.1. The skip mutation

We can now instantiate the generic human mutation in Definition 7 to the case in which a human skips one or more actions $a_i, a_{i+1}, \ldots$ in a subtrace $[a_0, \ldots, a_i, \ldots, a_n]^H$.

Definition 8 (Skip mutation)**.** *A skip* mutation

$$\mu^H_{skip} : tr \mapsto tr'$$

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* $k \geqslant 1$ *is the action that H executes immediately after the execution of* a_{i-1} *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 20 shows a human subtrace in which H skips the $Snd(H, \mathrm{sec}, GateIn, card)$ action in the Oyster ceremony (omitting constants for readability, 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. Note that Figure 20 also serves as an illustration for the need of matching and propagating mutations: if the human skips the $Snd(H, \mathrm{sec}, GateIn, card)$ action, then it is highly unlikely, albeit not impossible, that the *GateIn* will be able to send the reply that the human expects to receive next, so we might need to remove also the action $Rcv(H, \mathrm{sec}, GateIn, \langle card, ginID\rangle)$. We will return to this more formally, and with more details, later (e.g., in Section 5.1.1).

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 be combined to skip bigger "chunks" of the ceremony execution.

We describe the five *skip* mutations by showing their effect on the trace (2).

$$
\begin{array}{lcl}
Start(H, \langle GateIn, GateOut, card, bal \rangle), & & Start(H, \langle GateIn, GateOut, card, bal \rangle), \\
Snd(H, \mathrm{sec}, GateIn, card), & & \cancel{Snd(H, \mathrm{sec}, GateIn, card)}, \\
Rcv(H, \mathrm{sec}, GateIn, \langle card, ginID \rangle), & \xrightarrow{\mu^H_{skip(S)}} & Rcv(H, \mathrm{sec}, GateIn, \langle card, ginID \rangle), \\
Snd(H, \mathrm{sec}, GateOut, \langle card, bal(card), ginID \rangle), & & Snd(H, \mathrm{sec}, GateOut, \langle card, bal(card), ginID \rangle), \\
Rcv(H, \mathrm{sec}, GateOut, \langle card, bal(card)', finish \rangle) & & Rcv(H, \mathrm{sec}, GateOut, \langle card, bal(card)', finish \rangle)
\end{array}
$$

- -

$$
\begin{array}{cl}
 & Start(H, \langle GateIn, GateOut, card, bal \rangle), \\
 & Snd(H, \mathrm{sec}, GateIn, card), \\
\xrightarrow{\mu^H_{replace}} & Rcv(H, \mathrm{sec}, GateIn, \langle card, ginID \rangle), \\
 & Snd(H, \mathrm{sec}, GateOut, \langle card2, bal(card2), ginID \rangle), \\
 & Rcv(H, \mathrm{sec}, GateOut, \langle card2, bal(card2)', finish \rangle)
\end{array}
$$

- -

$$
\begin{array}{lcc}
 & \multicolumn{2}{c}{Start(H, \langle GateIn, GateOut, card, bal \rangle),} \\
 & \multicolumn{2}{c}{Snd(H, \mathrm{sec}, GateIn, card),} \\
\xrightarrow{\mu^H_{add} \cdot \mu^H_{replace}} & \multicolumn{2}{c}{Rcv(H, \mathrm{sec}, GateIn, \langle card, ginID \rangle),} \\
 & Snd(H, \mathrm{sec}, GateOut, \langle card, bal(card), ginID \rangle), & Snd(H, \mathrm{sec}, GateOut, \langle card2, bal(card2), ginID \rangle), \\
 & Rcv(H, \mathrm{sec}, GateOut, \langle card, bal(card)', finish \rangle) & Rcv(H, \mathrm{sec}, GateOut, \langle card2, bal(card2)', finish \rangle)
\end{array}
$$

Figure 20. Examples of mutations in the case of the Oyster ceremony

5.1.1. 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 $j \geqslant i+1$, which we call the *landing transition* (i.e., the transition where H lands after the "jump" he has made):[13]

$$\begin{array}{c}
\vdots \Sigma_1 \\
\mathsf{AgSt}(H,i,kn_i),\ Pre_i,\ Rcv(H,l_1,A_1,m_1) \rightarrow \\
\mathsf{AgSt}(H,i+1,kn_{i+1}),\ Post_{i+1},\ \cancel{Snd(H,l_2,A_2,m_2)} \\
\vdots [\![\Sigma_2]\!]^{\mu} \\
\mathsf{AgSt}(H,j,[\![kn_j]\!]^{\mu}),\ Pre_j,\ Rcv(H,l_3,A_3,[\![m_3]\!]^{\mu}) \rightarrow \\
\mathsf{AgSt}(H,j+1,[\![kn_{j+1}]\!]^{\mu}),\ Post_{j+1},\ Snd(H,l_4,A_4,[\![m_4]\!]^{\mu}) \\
\vdots [\![\Sigma_3]\!]^{\mu}
\end{array} \quad (3)$$

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; note that comments are written in italics and their start is denoted by ▹.

Algorithm 1 $\mu^H_{skip(S)}$: skip $Snd(H,l_2,A_2,m_2)$ in transition i, with landing transition j as in the trace (3)

```
1: Mutate transition $i$ to
       $\mathsf{AgSt}(H,i,kn_i),\ Pre_i,\ Rcv(H,l_1,A_1,m_1) \rightarrow$
       $\mathsf{AgSt}(H,i+1,kn_{i+1}),\ Post_{i+1}$
2: $[\![kn_j]\!]^{\mu} ::= kn_j = kn_{i+1}$   ▹ Since $\Sigma_2$, and thus $[\![\Sigma_2]\!]^{\mu}$, does not contain a transition in which $H$ receives new information
       ▹ There are two cases, depending on which message is sent by $A_3$, the original $m_3$ or its mutation $[\![m_3]\!]^{\mu}$
```

[13]For simplicity but without loss of generality, 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.

```
3: if $[\![\Sigma_2]\!]^\mu$ contains a transition with $Snd(A_3, l_3, H, m_3)$ in its conclusions then
4:     transition $j$ is the same as the original one in trace $tr$, i.e.
               $\mathsf{AgSt}(H, j, kn_j),\ Pre_j,\ Rcv(H, l_3, A_3, m_3) \rightarrow$
               $\mathsf{AgSt}(H, j+1, kn_{j+1}),\ Post_{j+1},\ Snd(H, l_4, A_4, m_4)$
5: else        ▷ $[\![\Sigma_2]\!]^\mu$ contains a transition with $Snd(A_3, l_3, H, [\![m_3]\!]^\mu)$ in its conclusion for some
   mutation $[\![m_3]\!]^\mu$ defined in Algorithm 2
6:     $[\![kn_{j+1}]\!]^\mu ::= kn_j \cup \{[\![m_3]\!]^\mu\} \cup Pre_j$
7:     build all $[\![m_4]\!]^\mu \in \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^\mu$
8:     mutate transition $j$ to
           $\mathsf{AgSt}(H, j, kn_j),\ Pre_j,\ Rcv(H, l_3, A_3, [\![m_3]\!]^\mu) \rightarrow$
           $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
```

The pseudo-code is hopefully quite explanatory, also thanks to the comments in the algorithms, but there are a couple of steps that deserve clarification. First of all, what does it mean that A_3 sends a mutation $[\![m_3]\!]^\mu$ 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 7 of Algorithm 1 we define $[\![m_4]\!]^\mu \in \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ of m_4, and then, for each of these mutations, we build all the corresponding mutated transitions j. Note also that we write $kn \cup Pre_l$ to mean the extension of some (possibly composite) knowledge kn with all messages generated freshly in Pre_l. Finally, note that we do not consider the case in which $[\![\Sigma_2]\!]^\mu$ 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. This would require Algorithm 1 to remove the corresponding receive $Rcv(H, l_3, A_3, m_3)$ or $Rcv(H, l_3, A_3, [\![m_3]\!]^\mu)$, a case that is already covered by the trace (4) and Algorithm 8 and Algorithm 9.

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

```
1: Consider the transition $next(i)$ in $\Sigma_2$ that immediately follows the mutated human transition $i$,
   i.e.
       $\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
       $\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$,
   where $A_s$ is one of the other agents and $l_p$ and $m_p$ are some channel and message as specified
   in $\Sigma_2$
2: remove $Rcv(A_2, l_2, H, m_2)$ from $next(i)$
```

```
3: $[\![kn_{x+1}]\!]^\mu ::= [\![kn_x]\!]^\mu \cup Pre_x$, where $[\![kn_x]\!]^\mu ::= kn_{x-1}$
4: build all $[\![m_p]\!]^\mu \in \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^\mu$
5: mutate the transition to
       $\mathsf{AgSt}(A_2, x, [\![kn_x]\!]^\mu),\ Pre_x\ \rightarrow$
       $\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^\mu),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^\mu)$
6: let $h ::= next(i)$
7: if the trace contains a transition $next(h)$ of the form
         $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
         $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$ then
8:     if this $next(h)$ is actually $H$'s landing transition $j$ already considered in Algorithm 1 then
9:         go to 7 with $h ::= next(h)$
10:    else
11:        replace $m_p$ with $[\![m_p]\!]^\mu$
12:        $[\![kn_{s+1}]\!]^\mu ::= [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$, where $[\![kn_s]\!]^\mu ::= kn_{s-1}$
13:        build all $[\![m_{p+1}]\!]^\mu \in \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated by $[\![kn_{s+1}]\!]^\mu$
14:        mutate the transition to
               $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
               $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
15:        go to 7 with $h ::= next(h)$
```

5.1.2. *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:

$$\begin{array}{c}
\vdots\Sigma_1 \\
\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \rightarrow \\
\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ \cancel{Snd(H, l_2, A_2, m_2)} \\
\vdots[\![\Sigma_2]\!]^\mu \\
\mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j,\ \cancel{Rcv(H, l_3, A_3, m_3)} \rightarrow \\
\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu) \\
\vdots[\![\Sigma_3]\!]^\mu
\end{array} \tag{4}$$

The pseudo-code for this case is in Algorithm 8 and in Algorithm 9; for the sake of readability, we give these and most of the following algorithms in the appendix.

5.1.3. *The skip mutation $\mu^{H}_{skip(R)}$*

In this case, H skips $Rcv(H, l_1, A_1, m_1)$ in transition i:

$$\begin{array}{c}
\vdots \Sigma_1 \\
\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ \cancel{Rcv(H, l_1, A_1, m_1)} \rightarrow \\
\mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^{\mu}),\ Post_{i+1},\ Snd(H, l_2, A_2, [\![m_2]\!]^{\mu}) \\
\vdots [\![\Sigma_2]\!]^{\mu} \\
\mathsf{AgSt}(H, j, [\![kn_j]\!]^{\mu}),\ Pre_j,\ Rcv(H, l_3, A_3, [\![m_3]\!]^{\mu}) \rightarrow \\
\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^{\mu}),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^{\mu}) \\
\vdots [\![\Sigma_3]\!]^{\mu}
\end{array} \tag{5}$$

The pseudo-code for this case is in Algorithm 10 and in Algorithm 11.

5.1.4. *The skip mutation $\mu^{H}_{skip(RS)}$*

In this case, H skips both $Rcv(H, l_1, A_1, m_1)$ and $Snd(H, l_2, A_2, m_2)$ in transition i:

$$\begin{array}{c}
\vdots \Sigma_1 \\
\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ \cancel{Rcv(H, l_1, A_1, m_1)} \rightarrow \\
\mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^{\mu}),\ Post_{i+1},\ \cancel{Snd(H, l_2, A_2, [\![m_2]\!]^{\mu})} \\
\vdots [\![\Sigma_2]\!]^{\mu} \\
\mathsf{AgSt}(H, j, [\![kn_j]\!]^{\mu}),\ Pre_j,\ Rcv(H, l_3, A_3, [\![m_3]\!]^{\mu}) \rightarrow \\
\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^{\mu}),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^{\mu}) \\
\vdots [\![\Sigma_3]\!]^{\mu}
\end{array} \tag{6}$$

The pseudo-code for this case is in Algorithm 12 and in Algorithm 13.

5.1.5. *The skip mutation $\mu^{H}_{skip(RSR)}$*

In this case, 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:

$$\begin{array}{c}
\vdots \Sigma_1 \\
\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ \cancel{Rcv(H, l_1, A_1, m_1)} \rightarrow \\
\mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^{\mu}),\ Post_{i+1},\ \cancel{Snd(H, l_2, A_2, [\![m_2]\!]^{\mu})} \\
\vdots [\![\Sigma_2]\!]^{\mu} \\
\mathsf{AgSt}(H, j, [\![kn_j]\!]^{\mu}),\ Pre_j,\ \cancel{Rcv(H, l_3, A_3, m_3)} \rightarrow \\
\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^{\mu}),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^{\mu}) \\
\vdots [\![\Sigma_3]\!]^{\mu}
\end{array} \tag{7}$$

The pseudo-code for this case is in Algorithm 14 and in Algorithm 15.

5.2. The add mutation

There are two different cases for this mutation: the human could

- send at any time any message that is created using the elements of one of the sets in $\mathcal{P}(kn) \setminus \{\emptyset\}$, where $\mathcal{P}(kn)$ is the *powerset* of the messages in the human's current knowledge *kn* (i.e., the finite set of all subsets of *kn* including $\{kn\}$), but we exclude the singleton containing the empty set as it does not make sense to send a message that belongs to the empty set (such a message does not exist),
- duplicate a send action.[14]

Note that if we allowed the human to send any message that he can build out of his current knowledge, then we would have to deal with an infinite set of options since, 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 *replace* mutation). In other words, since an agent's knowledge is never empty as it contains at the very least the agent's name, the closure of that knowledge under the generation and analysis rules will yield an infinite set. For instance, we can apply the pairing rule infinitely often to a single message term m to generate $\langle m,m\rangle$, $\langle m,m,m\rangle$, $\langle m,m,m,m\rangle$ and so on. In security protocol analysis, the ability of the attacker to generate infinitely many messages 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" [38], to manage the remaining infinite set of "answerable" messages). In other words, the attacker messages that cannot be answered by the other agents are simply excluded. In our approach, we cannot control the ability of the human user to generate, via the *add* mutation (and similarly for the *replace* mutation), infinitely many messages since our approach also generates mutations in the behavior of the other agents to match these human mutations, so the other agents will be able to respond to any human message. We thus cannot exclude a priori any of these infinitely many messages. Similar to [8], 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 the *powerset* $\mathcal{P}(kn)$ of *kn*, i.e., the finite set of all subsets of *kn* including $\{kn\}$ but excluding the singleton containing the empty set as it does not make sense to send a message that belongs to the empty set. We leave the investigation of other controlled notions of "sendable" messages to future work.

Definition 9. *An add mutation is a mutation*

$$\mu^H_{add} : tr \mapsto tr \times tr'$$

such that the original trace $tr = [a_0,\dots,a_{i-1},a_i,a_{i+1},\dots,a_n]$ *is run in parallel with the new, mutated trace* $tr' = [a_0,\dots,a_{i-1},[\![a_i]\!]^\mu,[\![a_{i+1}]\!]^\mu,\dots,[\![a_n]\!]^\mu]$*, 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* $A,m \in P$ *for a set* $P \in \mathcal{P}(kn_i) \setminus \{\emptyset\}$ *in the powerset of the messages in H's current knowledge* kn_i*, and* $[\![a_{i+1}]\!]^\mu,\dots,[\![a_n]\!]^\mu$ *are the mutations of the actions* $a_i,\dots,a_{i+k-1}$ *obtained by matching and propagating this mutation.*

[14]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. In contrast, the approach of [8] allows human agents to receive arbitrary messages but without considering mutations for the human or other agents.

Consider the beginning of the trace (2). The mutation μ^H_{add} mutates this to either

$$\begin{array}{c} \vdots \Sigma_1 \\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, M_1) \rightarrow \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ Snd(H, l_2, A_2, M_2) \\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, M_1) \rightarrow \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Snd(H, l, A, M) \\ \vdots [\![\Sigma_2]\!]^\mu \end{array} \tag{8}$$

for a new message M as described in Definition 9, or

$$\begin{array}{c} \vdots \Sigma_1 \\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, M_1) \rightarrow \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ Snd(H, l_2, A_2, M_2) \\ \mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, M_1) \rightarrow \\ \mathsf{AgSt}(H, i+1, kn_{i+1}),\ Snd(H, l_2, A_2, [\![M_2]\!]^\mu) \\ \vdots [\![\Sigma_2]\!]^\mu \end{array} \tag{9}$$

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

- sending at any time any message that the human knows can be combined with a *skip* mutation (e.g., to skip some steps of a ceremony and instead send 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 20.

Our tool implements this mutation as described in the pseudo-code in Algorithm 3 along with Algorithm 4 for the matching mutation.

Algorithm 3 *add* mutation μ^H_{add} as in the subtraces (8) and (9)

```
1: Add a transition at state $i$ built by either
2:     adding a $Snd(H, l, A, m)$ for some $l$, $A$ and $m \in P$ for $P \in \mathcal{P}(kn_i) \setminus \{\emptyset\}$, 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.
           $\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \rightarrow$
           $\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Snd(H, l, A, m)$
3:     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 4 Matching mutation for $\mu^{m(\mu^{H}_{add})}$

```
1: 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 3
2: considering the receiver $A_s$, take its $next(i)$ transition that immediately follows the mutated
   human transition $i$
3: 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 $A_s$, i.e.
       $\mathsf{AgSt}(A_s,s,kn_s),\ Pre_s,\ Rcv(A_s,l,H,m) \rightarrow$
       $\mathsf{AgSt}(A_s,s+1,kn_{s+1})$
4: 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
```

Note that in this matching mutation algorithm we allow the agent A to receive two copies of the message sent by the human, but do not duplicate the ensuing send action by A. This is in line with what we wrote above about the add mutation being best exemplified when combined with a replace mutation, and indeed this is what happens in our Oyster example, in which add&replace allows us to consider the case in which the human touches in with two cards, an Oyster card and a credit card as in Figure 20, and other similar cases. One could consider an alternative matching mutation algorithm in which A's send action is duplicated too and this is propagated accordingly, as well as other algorithms that mix different ways of adding and duplicating transitions.

5.3. 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). Like for the add mutation, we control the infinite set of message options by considering $\mathcal{P}(kn) \setminus \{\emptyset\}$, i.e., the finite set of all subsets of *kn* including *kn* but excluding the singleton containing the empty set, as it does not make sense to send a message which belongs to the empty set. Moreover, to simplify further, we restrict our attention to messages of the same type (this can be achieved thanks to the constants that we use to represent types, as we write in the definition and in the mutation algorithm below). Again, we leave the investigation of other controlled notions of "sendable" messages to future work.

Definition 10. *A replace mutation*

$$\mu^H_{replace} : tr \mapsto tr'$$

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 a set P in the powerset* $\mathcal{P}(kn_i) \setminus \{\emptyset\}$ *of the messages in H's current knowledge* kn_i *(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 trace (2):

$$\begin{array}{c} \vdots \Sigma_1 \\ \mathsf{AgSt}(H,i,kn_i),\ Pre_i,\ Rcv(H,l_1,A_1,m_1) \rightarrow \\ \mathsf{AgSt}(H,i+1,kn_{i+1}),\ Post_{i+1},\ Snd(H,l_2,A_2,[\![m_2]\!]^\mu) \\ \vdots [\![\Sigma_2]\!]^\mu \\ \mathsf{AgSt}(H,i+1,kn_{i+1}),\ Pre_{i+1},\ Rcv(H,l_3,A_3,[\![m_3]\!]^\mu) \rightarrow \\ \mathsf{AgSt}(H,i+2,[\![kn_{i+2}]\!]^\mu),\ Post_{i+2},\ Snd(H,l_4,A_4,[\![m_4]\!]^\mu) \\ \vdots [\![\Sigma_3]\!]^\mu \end{array} \qquad (10)$$

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 $[\![m_2]\!]^\mu \in \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ 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 5 along with Algorithm 6 for the matching mutation.[15] Note that in the trace (10) we do not have a landing transition as for the *skip* mutation in, e.g., the trace (2), since in the trace (10) Σ_2, and thus $[\![\Sigma_2]\!]^\mu$, does not contain other transitions of H; in other words, $j+1$ is the transition of H that immediately follows i and Σ_2, and thus $[\![\Sigma_2]\!]^\mu$ only contains one or more transitions by other agents.

For example, H could start the Oyster ceremony with one card *card*1 and complete it with another card *card*2, thus giving rise to two "incomplete journeys", as shown in Figure 20 and discussed in Section 7.1. For another example, see the SSO ceremony in Section 7.2.

Algorithm 5 *replace* mutation $\mu^H_{replace}$ as in the trace (10)

```
1: Build all transitions $i$ obtained by replacing $m_2$ either with each $[\![m_2]\!]^\mu \in \{(format(m_2))(m) \mid$
```

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

```
   $m \in submsg(m_2)\}$ or with each $[\![m_2]\!]^\mu$ that is in a non-empty set in the powerset of $H$'s current
   knowledge $kn_{i+1}$ but preserving types as specified by the corresponding constants, i.e., mutate
   transition $i$ to
           $\mathsf{AgSt}(H,i,kn_i),\ Pre_i, Rcv(H,l_1,A_1,m_1) \rightarrow$
           $\mathsf{AgSt}(H,i+1,kn_{i+1}),\ Post_{i+1},\ Snd(H,l_2,A_2,[\![m_2]\!]^\mu)$
   and for each of these transitions
2: if $[\![\Sigma_2]\!]^\mu$ contains a transition with $Snd(A_3,l_3,H,m_3)$ in its conclusions then  ▷ this means that the new message $[\![m_2]\!]^\mu$ has no influence on $m_3$
3:     $[\![kn_{i+2}]\!]^\mu ::= kn_{i+2}$, $[\![m_3]\!]^\mu ::= m_3$, $[\![m_4]\!]^\mu ::= m_4$ and transition $i+1$ is
           $\mathsf{AgSt}(H,i+1,kn_{i+1}),\ Pre_{i+1}, Rcv(H,l_3,A_3,m_3) \rightarrow$
           $\mathsf{AgSt}(H,i+2,kn_{i+2}),\ Post_{i+2},\ Snd(H,l_4,A_4,m_4)$
4: else  ▷ the new message $[\![m_2]\!]^\mu$ has some influence on $m_3$ and thus $A_3$ sends some $[\![m_3]\!]^\mu$, so we consider two cases depending on the choice of $[\![m_2]\!]^\mu$
5:     if $m_2$ was replaced with a $[\![m_2]\!]^\mu \in \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ then
6:         $[\![kn_{i+2}]\!]^\mu ::= kn_{i+1} \cup [\![m_3]\!]^\mu \cup Pre_{i+1}$
7:         build all $[\![m_4]\!]^\mu \in \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{i+2}]\!]^\mu$
8:         mutate transition $i+1$ to
           $\mathsf{AgSt}(H,i+1,kn_{i+1}),\ Pre_{i+1}, Rcv(H,l_3,A_3,[\![m_3]\!]^\mu) \rightarrow$
           $\mathsf{AgSt}(H,i+2,[\![kn_{i+2}]\!]^\mu),\ Post_{i+2},\ Snd(H,l_4,A_4,[\![m_4]\!]^\mu)$
9:     if $m_2$ was replaced with a $[\![m_2]\!]^\mu$ that is in a non-empty set in the powerset of $H$'s current knowledge $kn_{i+1}$ but preserving types as specified by the corresponding constants then
10:        $[\![kn_{i+2}]\!]^\mu ::= kn_{i+1} \cup [\![m_3]\!]^\mu \cup Pre_{i+1}$
11:        replace $m_4$ with each $[\![m_4]\!]^\mu$ that is in a non-empty set in the powerset of $H$'s current knowledge $kn_{i+2}$ but preserving types as specified by the corresponding constants
12:        mutate transition $i+1$ to
           $\mathsf{AgSt}(H,i+1,kn_{i+1}),\ Pre_{i+1}, Rcv(H,l_3,A_3,[\![m_3]\!]^\mu) \rightarrow$
           $\mathsf{AgSt}(H,i+2,[\![kn_{i+2}]\!]^\mu),\ Post_{i+2},\ Snd(H,l_4,A_4,[\![m_4]\!]^\mu)$
```

Algorithm 6 Matching mutation for $\mu^{m(\mu^H_{replace})}$

```
1: Consider the transition $next(i)$ that immediately follows the mutated human transition $i$, i.e.
           $\mathsf{AgSt}(A_2,x,kn_x),\ Pre_x,\ Rcv(A_2,l_2,H,m_2) \rightarrow$
```

```
        $\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p),$
    where $A_s$ is one of the other agents and $l_p$ and $m_p$ are some channel and message, and $m_2$ is
    replaced either by $[\![m_2]\!]^\mu \in \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ or by a $[\![m_2]\!]^\mu$ in a non-
    empty set in the powerset of $H$'s current knowledge $kn_{i+1}$ but preserving types as specified by
    the corresponding constants
 2: if $[\![m_2]\!]^\mu \in \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ then
 3:     $[\![kn_{x+1}]\!]^\mu ::= kn_x \cup Pre_x \cup [\![m_2]\!]^\mu$
 4:     build all $[\![m_p]\!]^\mu \in \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^\mu$
 5:     mutate the transition $next(i)$ to
            $\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, [\![m_2]\!]^\mu) \rightarrow$
            $\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^\mu),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^\mu)$
 6: else        ▷ $[\![m_2]\!]^\mu$ is in a non-empty set in the powerset of $H$'s current knowledge $kn_{i+1}$ but
    preserving types as specified by the corresponding constants
 7:     $[\![kn_{x+1}]\!]^\mu ::= kn_x \cup Pre_x \cup [\![m_2]\!]^\mu$
 8:     replace $m_p$ with each $[\![m_p]\!]^\mu$ in a non-empty set in the powerset of $A_2$'s current knowledge
    $[\![kn_{x+1}]\!]^\mu$ but preserving types as specified by the corresponding constants, and mutate the tran-
    sition $next(i)$ to
            $\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, [\![m_2]\!]^\mu) \rightarrow$
            $\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^\mu),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^\mu)$
 9: let $h ::= next(i)$
10: if the trace contains a transition $next(h)$ of the form
            $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
            $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$ then
11:     if this $next(h)$ is actually $H$'s transition $i+1$ already considered in Algorithm 5 then
12:         go to 10 with $h ::= next(h)$
13:     else
14:         if $[\![m_p]\!]^\mu \in \{(format(m_{p-1}))(m) \mid m \in submsg(m_{p-1})\}$ then
15:             $[\![kn_{s+1}]\!]^\mu ::= [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$, where $[\![kn_s]\!]^\mu ::= kn_{s-1}$
16:             build all $[\![m_{p+1}]\!]^\mu \in \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be gener-
    ated by $[\![kn_{s+1}]\!]^\mu$
17:             mutate the transition $next(h)$ to
                $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
                $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
18:         else    ▷ $[\![m_p]\!]^\mu$ is in a non-empty set in the powerset of $A_{s-1}$'s current knowledge but
```

```
    preserving types as specified by the corresponding constants
19:             $[\![kn_{s+1}]\!]^{\mu} ::= [\![kn_s]\!]^{\mu} \cup Pre_s \cup [\![m_p]\!]^{\mu}$, where $[\![kn_s]\!]^{\mu} ::= kn_{s-1}$
20:             replace $m_{p+1}$ with each $[\![m_{p+1}]\!]^{\mu}$ in a non-empty set in the powerset of $A_s$'s cur-
    rent knowledge $[\![kn_{s+1}]\!]^{\mu}$ but preserving types as specified by the corresponding constants, and
    mutate the transition $next(h)$ to
                $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^{\mu}),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^{\mu}) \rightarrow$
                $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^{\mu}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^{\mu})$
21:         go to 10 with $h ::= next(h)$
```

5.4. The neglect mutation

This mutation captures the fact that a human user may not adhere to one or more internal behaviors expected by the ceremony, e.g., the case in which the coach *Driver* neglects to execute a check on a particular field on a ticket during the Coach Service ceremony. This is defined by the mutation as the removal of one of more role-actions.

Definition 11. *A neglect* mutation

$$\mu^{H}_{action} : tr \mapsto tr'$$

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* $[a_0, \ldots, a_{i-1}, [\![a_i]\!]^{\mu}, [\![a_{i+1}]\!]^{\mu}, \ldots, [\![a_k]\!]^{\mu}, a_{k+1}, \ldots, a_n]$, *where* $\{a_i, a_{i+1}, \ldots, a_k\} = a$ *are the role-actions in the human transition rule* $prem \xrightarrow{a} conc$ *that has been mutated by the human into* $prem \xrightarrow{[\![a]\!]^{\mu}} conc$.

In the specific case that we are considering in this paper, the mutation of a Tamarin role-action in a transition rule $prem \xrightarrow{a} conc$ means either removing it from a or leaving it unchanged, but, in general, it could mean also modifying it by changing some of its parameters.

We cannot illustrate this mutation by means of the beginning of the trace (2) as we need to show the role-actions explicitly, but we can use the Coach Service ceremony and, in particular, consider the following mutation of the rule (D_2) that we gave in Figure 30:

$$[\mathsf{AgSt}(\mathit{Driver}, 2, \langle \mathit{kS}, \mathit{Customer}, \mathit{tknumber}, \mathit{date}, \mathit{dtime}, \mathit{from}, \mathit{to}\rangle)]$$

$$\xrightarrow{\substack{\mathit{Snd}(\mathit{Driver},\mathrm{sec},\mathit{Customer},\langle \mathit{tknumber},\text{‘}\mathit{ack}\text{’},\text{‘}\mathit{valid}\text{’}\rangle) \\ \mathit{Eq}(\mathit{tknumber},\mathit{tknumber}_{kS}), \\ \mathit{Eq}(\mathit{Customer},\mathit{Customer}_{kS}), \\ \mathit{Eq}(\mathit{price},\mathit{price}_{kS}), \\ \cancel{\mathit{Eq}(\mathit{date},\mathit{date}_{kS})}, \\ \mathit{Eq}(\mathit{dtime},\mathit{dtime}_{kS}), \\ \mathit{Eq}(\mathit{from},\mathit{from}_{kS}), \\ \mathit{Eq}(\mathit{to},\mathit{to}_{kS})}}$$

$$[\mathsf{AgSt}(\mathit{Driver}, 3, \langle \mathit{kS}, \mathit{Customer}, \mathit{tknumber}, \mathit{date}, \mathit{dtime}, \mathit{from}, \mathit{to}\rangle,$$
$$\mathsf{Out}_{sec}(\mathit{Driver}, \mathit{Customer}, \langle \mathit{Customer}, \mathit{tknumber}, \mathit{date}, \text{‘}\mathit{ack}\text{’}, \text{‘}\mathit{valid}\text{’}\rangle))]$$

This mutated rule reflects a mistake of the *Driver* of the coach who forgets to control that the date printed on the e-ticket shown by the *Customer* is, in fact, the original date of the journey for which it was bought. The removal of this role-action will entail that all the possible traces where this check is not valid are considered during the analysis of the security goal, whereas these traces would have been excluded in presence of the check, thus preventing the forging attack that we discussed in Section 2.3.

Our tool implements the *neglect* mutation as described in the pseudo-code in Algorithm 7. Note that this mutation is not matched. To clarify this, let us consider the possible cases:

- since we defined a strong correlation between some of the events in Tamarin and some facts (cf. Section 4.3), the *neglect* mutation is not allowed to remove the Tamarin role-actions that correspond to these facts because otherwise the resulting rule wouldn't be well-formed (moreover, the removal or modification of $\mathsf{Out}_l(A,P,m)$ and $\mathsf{In}_l(P,A,m)$ facts are already covered by the other mutations, cf. Definition 8);
- the *neglect* mutation is also not allowed to remove a Tamarin role-action that is necessary for a security goal because otherwise we would obtain a ceremony that is not executable;[16]
- if the *neglect* mutation removes one or more Tamarin role-actions that entail some restriction on the set of traces (e.g., an equality check $\mathit{Eq}(\mathit{date},\mathit{date}_{kS})$ like in the example considered here), then the actual Tamarin restriction is still valid as it is a generic property of the operator (e.g., of the equality operator *Eq*) but it simply will not be applied (as $\mathit{Eq}(\mathit{date},\mathit{date}_{kS})$ is not present anymore) and the X-Men tool will thus analyze more traces, possibly including some traces that contain an attack based on the mutation;
- in the role scripts of the other agents there is nothing that corresponds to the events of another agent (so the other agents will not notice if one or more Tamarin actions of the human are removed).

Algorithm 7 *neglect* mutation μ^H_{action}

```
1: Build all transitions $i$ obtained with $[\![a]\!]^{\mu}$ by removing one or more Tamarin role-actions accord-
   ing to the powerset generated from the original Tamarin role-action set $\{a_i, a_{i+1}, \ldots, a_k\} = a$,
```

[16]Another possibility would be to extend the mutation to deal with the removal of the Tamarin role-actions necessary for a security goal, but we leave this to future work.

```
    i.e.
        $\mathsf{AgSt}(H, i, kn_i),\ Pre_i, Rcv(H, l_1, A_1, m_1) \xrightarrow{[\![a]\!]^\mu}$
        $\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1},\ Snd(H, l_2, A_2, m_2)$
2:  let $h ::= i$
3:  if the trace contains a transition $next(h)$ of the form
        $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \xrightarrow{a'}$
        $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$ then
4:      if $A_s = H$ and $a'$ contains the same one or more Tamarin role-actions that have been removed then
5:          remove the same occurrences of one or more Tamarin role-actions from $a'$ in $next(h)$ to replace $next(h)$ with
                $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_y,\ Rcv(A_s, l_p, A_{s-1}, m_p) \xrightarrow{[\![a']\!]^\mu}$
                $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$
6:          go to 3 with $h ::= next(h)$
7:      else
8:          go to 3 with $h ::= next(h)$
```

Note that in this algorithm if the human neglects a check at some point, and thus the corresponding role-action is removed, then the algorithm also removes the same check (and role-action) in all the following transitions in which the check appears (if at all, since in fact it will depend on the ceremony, and on how the modeler models it, whether the check is carried out only once or repeated in many transitions). This models the fact that the human neglects that check altogether. An interesting variant that could be considered in the future is the case in which the role action is removed in only one transition but kept in all subsequent ones, thereby modeling a momentary lapse.

6. X-Men: A Tool for the Generation of Mutated Specifications Based on Human Behaviors

In this section, we describe the X-Men Tool and how it generates the mutated models that are then analyzed using Tamarin.

As we remarked in the introduction, X-Men fully automates the workflow of our approach, except for the green box "Security Analyst Checks", which is carried out by hand by the analyst (cf. Figure 1). As shown in Figure 21, X-Men fully automates the process of generation and analysis of mutated security ceremony models that are then automatically used as input to Tamarin. X-Men can be used with human mutations only (without matching, e.g., as in the case of the Coach Service ceremony), or it can be used with matching mutations that are propagated to create an executable trace that can be analyzed in search for attacks (as for our two other case studies).

Our approach works in three fully-automated phases as shown in Figure 2:

(1) the preprocessing phase, which prepares the specification file for the mutations,

(2) the mutation phase, which generates the mutated specifications, and
(3) the analysis phase, which invoke Tamarin to carry out the analysis and outputs a report of the results.

6.1. The preprocessing phase

The entire execution of the X-Men Tool is managed by two scripts, *Wolverine* and *Xavier*, which are written using the Python programming language and which collaborate with each other to make it possible to generate mutated models and analyze them automatically. The starting point of the entire process is the Wolverine script which, in Figure 2, is shown as "*Slicer/Joiner*": Wolverine's primary functions are to take care of the slicing, and consequently the joining, of the input ceremony models.[17] Given a model written using the standard Tamarin syntax and order of rules, the slicing function within Wolverine splits it into three different parts and produces in output three different files, one file per part.[18] Users of the X-Men Tool must use specific "delimiters" (defined in [25]) to identify these three parts, which are:

(1) a first part that contains everything from the beginning of the model until the channel rules (included),
(2) a second part that contains the functions and all the agent rules,
(3) a third part that contains the restrictions and lemmas.

Once the three files have been generated, Wolverine invokes the X-Men Tool to carry out the mutation phase.

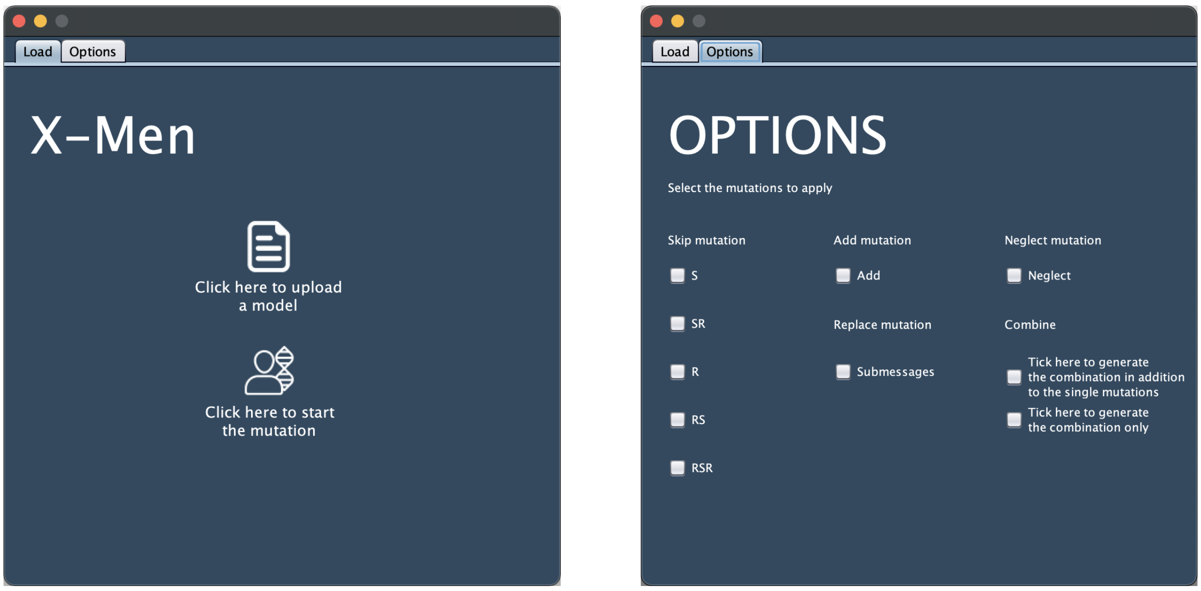
6.2. The mutation phase

The mutation phase is managed by the X-Men Tool, which is represented in Figure 2 by the black rectangle with a yellow-black "X" containing the *behavioral patterns library*. X-Men is written using the *Java programming language* following the Model–View–Controller software design pattern. To be able to read model files written in the Tamarin syntax, X-Men takes advantage of another component called *ANTLR* [40], which stands for **AN**other **T**ool for **L**anguage **R**ecognition. ANTLR is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It is widely used to build tools and frameworks, and its role is to generate from a grammar a parser that can build and walk through parse trees. We have written the grammar that is the starting point for ANTLR following the syntax of the Tamarin grammar in [36].

Once Wolverine has invoked X-Men, a *Graphical User Interface (GUI)* appears as shown in Figure 21a. The GUI is quite clear and simple, and guides a security analyst through the few steps needed to generate the mutated models. In fact, when opened, the GUI allows the security analyst only to upload a model. Once a model has been uploaded, a new interface tab allows the security analyst to select which mutations he wants to apply. The new tab, shown in Figure 21b, allows the security analyst to select the four mutations (*skip*, *replace*, *add*, *neglect*), their variants (e.g., *S*, *SR*, *R* for *skip*, *submessages* or *type*

[17]We named the script after the mutant Wolverine, a member of the X-Men who possesses retractable adamantium claws that can slice through (almost) anything.

[18]In addition to these files, another file is created to preserve the original model file as a backup copy. This file will be restored at the end of the execution of the X-Men Tool to allow the security analyst to execute the tool as many times as desired with minimum hassle when dealing with modifications of the original files.


Load
Options
X-Men
Click here to upload a model
Click here to start the mutation
Load
Options
OPTIONS
Select the mutations to apply
Skip mutation
S
SR
R
RS
RSR
Add mutation
Add
Replace mutation
Submessages
Neglect mutation
Neglect
Combine
Tick here to generate the combination in addition to the single mutations
Tick here to generate the combination only

(a) The main panel of the X-Men Tool

(b) The mutations panel in the X-Men Tool

Figure 21. The GUI of the X-Men Tool

for *replace*, etc...) and their combinations, and any other mutation that will be defined in X-Men's library of behavioral patterns in the future.

Once the security analyst has selected the mutations he wants to apply to the model, he can execute the generation of the mutated models by clicking on the button that is shown in Figure 21a. Once X-Men has terminated the generation of the mutated models, Wolverine takes back the control of the process and performs the joining of each mutated model created from part (2), which includes the functions and all the ceremony agent rules, with the other two parts, part (1) containing the initial definitions and the channel rules and part (3) containing the restrictions and lemmas. Once the joining has terminated, the X-Men Tool completes the generation of the mutated models and feeds them to the analysis phase.

6.3. The analysis phase

As we discussed above and as shown for our three case studies in Table 1, X-Men generates a large number of mutated models. We have implemented the Python script Xavier to tackle this issue.[19] In order to simplify the analysis of such a large number of mutated models, Xavier automates the analysis process by fetching all the mutated models, inputting them into Tamarin and then, for each model, reporting

- whether one or more properties have been falsified (and thus a vulnerability identified),
- whether Tamarin has timed out before finding a proof (we have set a timeout of 10 minutes, which is usually a long enough time for Tamarin to produce an output, but of course a longer timeout could be set, especially for large and complex ceremonies), or
- whether the model has been successfully verified.

[19]We named the script after the mutant Professor X (Charles Francis Xavier), the founder and leader of the X-Men.

Table 1 summarizes the reports generated by Xavier for our three case studies. The first column shows which mutation is considered, the second column shows which version of that specific mutation is considered, the third column shows if the attacker is enabled. The following columns display the results for each case study: how many mutated models were generated by X-Men, how many mutated models had a security property (or more properties) falsified by Tamarin, how many models made Tamarin time out, and how many models were verified by Tamarin.

Table 1

Mutated models generated by X-Men
("Yes" indicates that the attacker is activated, "No" indicates that the attacker is not activated, "-" indicates that that mutation is not considered in that case study)

Mutation		**Attacker**	**Case Study**											
			Oyster				SSO				Coach Service			
			Models				**Models**				**Models**			
			Generated	Falsified	Timeout	Verified	Generated	Falsified	Timeout	Verified	Generated	Falsified	Timeout	Verified
skip	S	No	2	1	0	1	-	-	-	-	-	-	-	-
	SR	No	3	2	0	1	-	-	-	-	-	-	-	-
	R	No	2	1	0	1	-	-	-	-	-	-	-	-
	RS	No	1	0	0	1	-	-	-	-	-	-	-	-
	RSR	No	1	1	0	0	-	-	-	-	-	-	-	-
replace	submessage	Yes	-	-	-	-	255	77	178	0	-	-	-	-
	type	No	3	1	0	2	-	-	-	-	-	-	-	-
add		No	92	48	2	42	-	-	-	-	-	-	-	-
add&replace	submessage	Yes	-	-	-	-	255	0	255	0	-	-	-	-
	type	No	3	2	0	1	-	-	-	-	-	-	-	-
neglect		Yes	-	-	-	-	-	-	-	-	127	95	0	32

It is important to note that, regardless of the performance of the X-Men Tool in the generation of mutated models, the overall performance of the entire analysis process depends on the actual performance of Tamarin and its prover, in addition to the actual performance of the computer on which X-Men is run. As is commonly the case in security protocol analysis, the security analyst will need to carry out an inspection of those models for which Tamarin timed out. Moreover, even though X-Men automates the entire process, the security analyst will need to inspect the models for which Tamarin identified vulnerabilities in order to check whether these are vulnerabilities that apply also to the original ceremony and whether the mutations are representative of interesting real-life scenarios. This manual analysis is made easier by Tamarin's graphic mode, which displays attack traces in a graphical and thus human-friendly way.

As we remarked above, this need for the manual intervention of a security analyst is similar to what happens with mutation-based testing approaches [19–24], and we leave for future work both extensions that would allow for more automation of the manual inspection and extensions that would allow X-Men to generate test cases from the attack traces and to apply them on the ceremony implementation.

7. Analysis of the three case studies

In this section, we show how our formalization can be used effectively for finding attacks that are due to the (mis-)behavior of human agents in security ceremonies. As proof-of-concept, we have applied our tool X-Men to three case studies, the Oyster ceremony, the Single Sign-on ceremony and the Coach Service ceremony. As remarked above and shown in Table 1, X-Men generated a large number of mutated models for these ceremonies. These mutated models are then automatically analyzed by Tamarin thanks to Xavier activating the attacker rules when necessary (for the SSO and the Coach Service case studies but not for the Oyster one). The next three subsections summarize the results of the analyses of the three case studies, and in Section 7.4 we then discuss how to exploit the results of an analysis of a security ceremony.

7.1. Analysis of the Oyster Ceremony

Table 2 shows some of the attacks found with the models obtained by applying the mutations skip, replace and add&replace to the Oyster ceremony. The table does not show the results pertaining to the 92 models generated by the add mutation: the report generated by the analysis indicates that 42 of these models are verified and 2 cause Tamarin to time out, whereas 48 models are falsified and require a check for false positives by the security analyst. We carried out that check and concluded that all these 48 falsified models are not realistic attacks as they represent cases in which the human agent systematically tries all the possible combinations of parameters without following the logic behind the ceremony. As we will discuss in more detail below, although many generated models can be excluded automatically as they are verified, there are still many models that are falsified (or time out) for which an inspection by the security analyst is required and it will be useful to provide additional automated support for this inspection. While it will be challenging to fully automate the inspection, we are currently working at devising techniques to identify common features among such models and thus group them into a small(er) equivalence classes, so that the analyst would then need to consider only one model per class. Distinguishing the falsified models that require matching mutations (like all these 48 models do) from those that do not require matching already provides some very initial but useful classification.

The column "Mutated model" of Table 2 lists the file identifier of the generated file (as used at [25]) 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 Section 4.4, we have used Tamarin to check the *functional* goal (a.k.a. *executable* goal) that the mutations did not create models that are not able to complete the message passing terminating with the last rule. All the models considered in the table passed this check.

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

Attack #1. The MSC of the attack (Figure 22) 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 *ginID*

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)

Mutation			**Mutation details**	**Explanation**	**Goal**			
		Mutated model			complete journey (*GO1*)	same card (*GO2*)	card clash (*GO3*)	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*)	×	×	×	●

on the card. The security goal *GO1* is not verified, entailing what TfL calls an *incomplete journey* as mentioned in Section 2.1, 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, and still do. For instance, a passenger might have two Oyster cards in her pocket and confuse them, or the passenger might use Apple/Google pay (cf. Section 2.1) 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

Table 2

(Continued) 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)

Mutation			**Mutation details**	**Explanation**	**Goal**			
		Mutated model			complete journey (*GO1*)	same card (*GO2*)	card clash (*GO3*)	functional
	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	×	×	×	●

and is used by TfL as the identifier for a single journey.

Attack #3. The MSC in Figure 23 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. Section 2.1). 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 found 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. Hence, in this case, the matching mutations represent the concrete behavior of the implementation of the TfL system. While these attacks are, to some extent, known to TfL (cf. their warnings in Figure 5) 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 find them automatically (based on an analysis of the specification, rather than out of observations in practice). Other attacks might be found by considering other goals or other mutations. Moreover, in the style of model-based testing (see the end of Section 3), it is possible to use

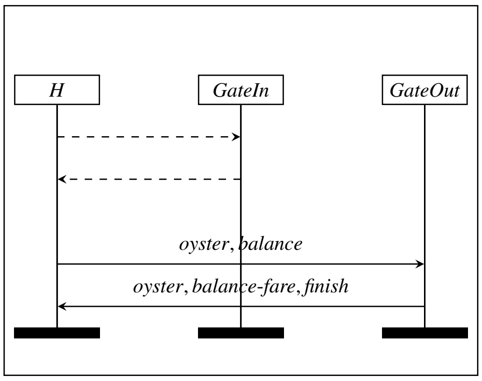


Figure 22. Attack that represents the Incomplete journey scenario for the Oyster ceremony

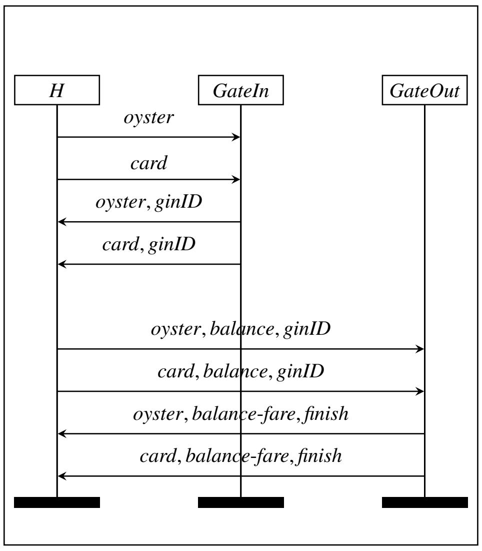


Figure 23. Attack that represents the Card clash scenario for the Oyster ceremony

the attack traces to generate concrete test cases to be executed on the code of the ceremony (if that is available).

7.2. Analysis of the SSO Ceremony

We have specified SSO as a ceremony in X-Men, considering what would happen if *SP* was played by the attacker and *IdP* was 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 as in Listing 4 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 [18]. Our mutations, among other things, capture such possible specification-implementation deviance.

Table 3
The attack found on the models obtained using the mutations applied to the Single Sign-on ceremony
("✓" indicates that an attack has been found, "×" indicates that no attack was found, "●" indicates that the functional goal is verified)

Mutation			**Mutation details**	**Explanation**	**Goal**	
		Mutated model			injective agree	functional
SSO		M0	Replace the message of *IdP*$_1$ with a submessage of the same message	The message is sent with the *Client* and the *IdP* identifier only (any information about the service provider *SP* is removed)	✓	●

Note that, in fact, SSO is a security protocol that does not involve human agents; in particular, the IdP is not a human, but in other contexts humans might indeed serve as identity providers and they might make mistakes like the ones we consider here, so we believe this example to be quite useful to illustrate this issue. When we were looking for a case study that would allow us to show the importance of the replace mutation, we choose the SSO example since Google engineers had implemented the protocol by replacing the original message in the standard specification with a submessage. In [18], the attack was found by human analysts (Armando et al.) inspecting the implementation and writing a specification that would represent the mistake done by the engineers. In a sense, Armando et al. had perceived that removing some fields from the messages might lead to attacks and they carried out an automated analysis to validate their intuition. We decided to use the SSO example to illustrate that, in contrast, our approach allows us to avoid having to rely on intuition and instead directly start from the actual specification and then consider an admittedly large number of mutations that however includes the one that gives rise to the attack. We could have looked for other case studies, but we wanted to show that our approach would have allowed security analysts to find the possible attack starting from the specification as it was written in the standard.

7.3. Analysis of the Coach Service Ceremony

The specification of the Coach Service ceremony includes:

- a phase in which the customer buys a genuine e-ticket (this phase is needed to obtain a first e-ticket that will be modified later, for other journeys) and
- a ticket inspection phase performed by a driver of a coach service.

We have specified the Coach Service as a ceremony in X-Men, considering what would happen if *Customer* was played by the attacker and *Driver* was played by a human, and the *neglect* mutation generates a new mutated rule in which the check of the genuineness of the fields is compromised (e.g., the check of the date).

The decision to use the attacker rules is due to the following facts:

- the neglect mutation focuses on showing a mistake caused by not carrying out a check rather than the creation of some messages,
- the creation of a forged ticket by the *Customer* goes beyond the simple application of a replace mutation (and is, in a sense, on a higher level than the other mutations),

- the use of another mutation of the ceremony that affects a different actor, in this case a mutation of the *Customer* in addition to the *neglect* mutation that affects the *Driver*, would require the two mutations to interact with each other, which our approach can cope with, although we leave a detailed formalization of such a situation for a future work.

We considered the scenario in which an attacking *Customer* reuses an e-ticket bought for a previous journey, modifying the validity date *date* into *date′*. The *neglect* mutation, as anticipated in Section 5.4, removes the role-action $Eq(date', date_{kS})$ from the rule (D_2) of the *Driver*. The consequence of the removal of this equality check is that Tamarin will consider traces that were not considered before (as the restriction prevented them from taking place). We fed our specification into Tamarin, which proved the existence of the attack in which the forged e-ticket can be accepted as valid by the *Driver* and the *Customer* is admitted to use the coach service, as shown in Figure 24.

Table 4
The attack found on the models obtained using the mutations applied to the Coach Service ceremony
("✓" indicates that an attack has been found, "×" indicates that no attack was found, "●" indicates that the functional goal is verified)

Mutation			**Mutation details**	**Explanation**	**Goal**	
		Mutated model			legit journey	functional
neglect		M0	An equality action is removed	The driver *D* does not check the validity of the date on the ticket	✓	●

7.4. *How to exploit the results of the analysis of a security ceremony*

We now briefly discuss how to exploit the results of the analysis of a security ceremony carried out using our approach. Similar to what happens for the formal analysis of security protocols, if the tool terminates and verifies the model under consideration, then we can provide a security guarantee. In the case of security ceremonies, our approach allows us to provide such guarantee not only for the original ceremony but also for its mutations for which Tamarin's analysis terminates with a proof. To some extent, this applies also when the analysis times out, although in that case, like in the case of a non-terminating analysis of a security protocol, an intervention of the security analyst is needed since the tool has not been able to give a definitive answer.

If the analysis of a security protocol model instead terminates with the discovery of an attack, typically the attack trace can be used to distill a fix to the protocol specification; one would also usually wish to check whether the attack on the model also applies for the concrete implementation (assuming that it is available), so the attack trace can also be used to devise test cases for the implementation (see, e.g., [23, 24, 41]). The same applies in the case of our mutated ceremonies when the model did not need to match and propagate mutations. Then, we can use the attack trace to distill a fix and generate test cases. If the model contains matching mutations, then the security analyst should check whether the matching of mutations yields a false positive that makes little sense in real life or whether the attack is a real attack (and then one would want to generate test cases similar to what is done in mutation-based testing).

In both these cases (real attack or false positive), given the presence of human users, one can also think of using the attack trace to distill recommendations and guidelines for the users of the ceremony

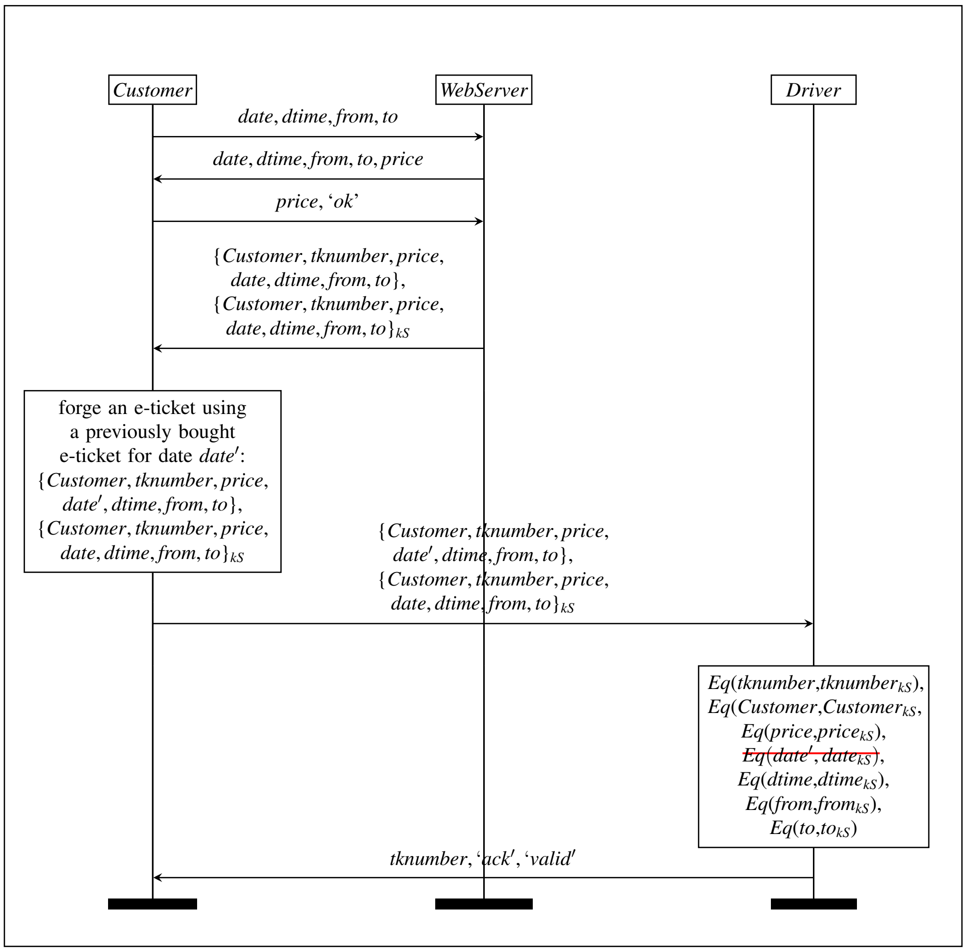

Customer
WebServer
Driver
date, dtime, from, to
date, dtime, from, to, price
price, 'ok'
{Customer, tknumber, price, date, dtime, from, to}, {Customer, tknumber, price, date, dtime, from, to}kS
forge an e-ticket using a previously bought e-ticket for date date': {Customer, tknumber, price, date', dtime, from, to}, {Customer, tknumber, price, date, dtime, from, to}kS
{Customer, tknumber, price, date', dtime, from, to}, {Customer, tknumber, price, date, dtime, from, to}kS
Eq(tknumber,tknumberkS), Eq(Customer,CustomerkS, Eq(price,pricekS), ~~Eq(date', datekS)~~, Eq(dtime,dtimekS), Eq(from,fromkS), Eq(to,tokS)
tknumber, 'ack', 'valid'

Figure 24. Attack on the Coach Service ceremony (for e-tickets): the *Customer* uses a forged e-ticket and the *Driver* validates it

so that they interact with it in a way that does not endanger security. After noticing the issues caused by the unexpected use of the Oyster/credit cards at the in/out gates, the posters in Figure 5 were hung in the London underground to remind users about the expected use of the cards, and one could use our approach to distill similar recommendations from a formal analysis before the system is actually deployed and the issues observed in practice. Similarly, one could use the attack traces as a basis to devise specific training for the ticket inspectors of the coach ceremony (as well as for humans acting as identity providers like in the SSO example), pointing out which behaviors and mistakes will lead to attacks. The rules of [8] that restrict how the human can deviate from the protocol specification are a good example for such guidelines. In future work, we aim to investigate if and how recommendations and guidelines could be generated (semi-)automatically from the analysis of security ceremonies and their mutations (similar to the generation of test cases), and how they could be communicated to human users in an effective way. To that end, we plan to exploit also our works on how to provide security explanations to laypersons [42–45] and on how to beautify security ceremonies [46–48] and thus make their secure use more appealing to human users.

8. Related Work

In this section, we discuss related work, expanding on the discussions that we have already given in previous sections. We compare our approach with [8], which is the most closely related work, but we also consider research that has inspired our work or that might provide interesting future synergies.

In [6], 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 [1], 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 [10, 49] 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. [8] 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 and there are some affinities, there are fundamental technological, methodological and philosophical differences between our approach and that of [8]. The four main differences are the following ones.

First, there are possible human behaviors that we consider that [8] does not. For instance, they consider a rule that allows humans to send "controlled" messages that are in their current knowledge. This overlaps with our add mutation and, to some extent, with the replace mutation, but their approach does not have an exact equivalent for the skip mutation nor for the neglect mutation. They can model a situation in which a human sends a message that he was meant to send much later in a ceremony, and that amounts indirectly to a form of skip, but our skip mutations cover a more general landscape. The approach of [8] includes an $ICompare(H, tag)$ predicate that "states that whenever the human H receives a message with tag tag, he compares it to the message associated with the same tag in his initial knowledge. The effect of this predicate, in combination with the untrained human rules, is that the human agent either ignores the entire message or verifies the tagged subterm." This is different from our neglect mutation, in which the human neglects to adhere to one or more internal behaviors expected by the ceremony, such as neglecting to carry out an internal action that is visible only to the agent itself, like a check on the contents of a message.

Second, Basin et al. 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. Fundamentally, as is standard in security protocol analysis, their approach takes the point of view of the Dolev-Yao attacker, in the sense that they look for what the Dolev-Yao attacker can do when human agents make the mistakes that they model in their approach. We do not take this point of view, but instead, in a sense, take the one of the human agent as we consider whether attacks can occur due to human mutations even in the absence of the Dolev-Yao attacker, as we do for instance in the case of the Oyster case study. This is not just technologically different (as the modeling is different) but also philosophically as via the mutations the human agent "forces" the attack on herself even in the total absence of an attacker (as in the Oyster example). We also include the possibility in which the Dolev-Yao attacker might be present and exploit the human's mistakes, as in the case of the SSO and Coach Service examples. In fact, the Coach Service ceremony also lends itself well to consider two humans, a customer who attacks by replacing some fields of a ticket and a ticket inspector who neglects to carry out all checks. While our mutations could possibly be modeled by restricting or extending the Dolev-Yao behavior, this would make us lose the philosophical difference that our approach works also in the absence of the attacker.

Third, while our approach allows a security analyst to consider only human mutations like [8], we formalize also matching and propagating mutations. Many of the human mistakes that we consider, if left to themselves, would not lead to an attack as the ceremony analysis would not terminate. As we discussed earlier, by matching and propagating the mutations, we investigate how they could lead to attacks, thus providing a wider (and in a sense also deeper) analysis. It is thanks to these mutations that X-Men is able to (re-)discover automatically the attacks on Oyster, SSO and Coach Service. If we did not have these mutations, then our approach, like a Dolev-Yao-based one that only considers the original specification even in the presence of human mistakes, would not be able to discover many of them as the ceremony execution would not terminate. Even if we took the Dolev-Yao-centric approach, we would still need to extend the Dolev-Yao attacker to be able to match the mutation (e.g., to be able to receive any message, in addition to being able to send any message). So, our approach, in contrast to that of [8] and instead similar to mutation-based testing, is not sound in the sense that we get false positives that the security analyst should check to identify attacks that would not exist if, e.g., the specification was implemented as originally specified since then the other agents would not respond to the human mutations. In return, however, our approach also encompasses the cases in which the original specification would allow them to respond, and we do so for more possible human mistakes.

Fourth, there are features of the approach of [8] that our approach does not yet capture. For instance, the approach of [8] allows human agents to receive arbitrary messages (with a certain structure). Since we only consider mutations initiated by a human agent, 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. We will consider adding this case in the future.

Moreover, the rules of [8] that restrict how the human can deviate from the protocol specification provide guidelines for users to interact with a protocol in a secure way. As we remarked above, in future work we plan to investigate how to extend our mutation-based approach to generate recommendations and guidelines, and how they could be communicated to human users in an effective way.

Another quite closely related work is that of Bella, Giustolisi and Schürmann in [50], which was partly inspired by our previous joint work [51] as well as the preliminary version of this paper [17]. In contrast to our approach, all technical components in their approach are assumed to behave as intended, and they consider explicitly distributed and interacting human threats, so that every human may misbehave for his personal sake, without any fixed prescription to collude with others, yet may directly favor someone else. There are also differences in the formal modeling of security ceremonies, since they employ epistemic modal logic, which allows them to formalize ceremonies, threats, and properties in a way that many readers will find more intuitive than when using Tamarin's constructs, but this comes at the cost of an extra effort for their encoding into the prover of choice (Tamarin or another tool).

We have also joined forces with Bella and Giustolisi in previous works, in particular [51], in which we have given a systematic definition of an encompassing method to build the full threat model chart for security ceremonies from which one can conveniently reify the threat models of interest for the ceremony under consideration. To demonstrate the relevance of the chart, we formalized this threat model using Tamarin and analyzed three real-life ceremonies that had already been considered, albeit at different levels of detail and analysis, in the literature: MP-Auth [7], Opera Mini [47], and the Danish Mobilpendlerkort ceremony [31]. The full threat model chart suggested some interesting threats that had not been investigated although they are well worth of scrutiny. In particular, we found out that the Danish Mobilpendlerkort ceremony is vulnerable to the combination of an attacking third party and a malicious phone of the ticket holder. The threat model that leads to this vulnerability had not been considered before and arose thanks to our charting method. We are currently working at combining the threat model chart with the mutation-based approach that we presented in this paper, which we believe will help us identify novel vulnerabilities but also, we hope, reduce the number of mutations that need to be considered.

Other related approaches are those of [11, 12, 15, 52, 53]. Similar to [8], Curzon et al. [11] 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 [12] 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* [54]. Their approach provides only a preliminary formalization without a security analysis.

Beckert and Beuster [52] 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 [53] 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 [15] 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 [11, 12, 15, 52, 53], 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.

9. Conclusions

At the end of this long paper, let us take stock, and summarize the main points and discuss some future work. Our approach allows us to treat humans as first-class actors in security ceremonies as they should be considered to be, since it is not enough to take the "black&white" view of security protocol analysis, in which there is a Dolev-Yao attacker (the black agent) against a set of honest agents (the white agents). It is namely not enough to model human users as "honest processes" or as attackers, because they are neither, and we need to model the behavior of human users of ceremonies as "shades of gray". Our approach allows us to model, by means of mutations, some of the ways humans interact with the other agents, their behavior and the mistakes they may make, independent of attacks and, in fact, possibly independent of the presence of an attacker since attacks may occur even without the presence of an attacker.

Our approach does, however, come at a price, explicited mainly by the large amount of mutations that are generated and by the need for the security analyst to intervene with a manual inspection to check for potential false positives or in case Tamarin times out. While the latter is a common problem in the analysis of security protocols/ceremonies, where many of the tools do not terminate as the search space is infinite in the presence of a Dolev-Yao-style attacker, the former is an issue akin to what happens in mutation-based testing. As future work, we plan to investigate how to improve the efficiency of our approach by

- reducing the number of generated mutated models (e.g., by identifying isomorphic models),
- automatically or at least semi-automatically checking whether attacks are real or not (thereby reducing the effort required of the security analyst in the green box in Figure 1).

We are also investigating how to link our formal analysis to mutation-based testing by generating test cases out of the attack traces, which will allow us to test concrete implementations of the ceremonies.

In Section 7.4, we have discussed how to exploit the results of the analysis of a security ceremony carried out by means of X-Men and in there and in the rest of the paper, we have already mentioned a number of directions for future work. We believe that the most interesting ones are the following ones.

Our current mutations are restricted by a number of constraints that we have imposed to be able to manage them efficiently, such as constraints on the types and formats of the messages. We plan to improve them by weakening of some of these constraints, say to consider other controlled notions of "sendable" message or the case in which the right message is composed in a wrong format.

We also plan to extend X-Men's library of behavioral patterns with other mutations, which will hopefully allow us to identify novel vulnerabilities in the ceremonies that we considered here and in the ones that X-Men will be applied to in the future. For instance, in this paper we do not model the fact that a human might forget some information as we follow the standard approach in which the knowledge of the agents increases monotonically, be they "machines" or humans. Forgetting could be modeled by a fifth human mutation, which would remove terms from the current knowledge of a human and thus limit the number of messages that the human can send at that stage. For example, this could be useful to model ceremonies for password recovery.

In addition to improving the four mutations considered here and to considering new mutations, we believe that it will be interesting to formalize complex combinations of mutations, in the spirit of the add&replace that we employed in this paper. To that end, it will be useful to attempt to prove compositionality results, e.g., showing under which conditions the analysis of composed mutations can be decomposed into the simpler analysis of the individual mutations (similar to what can be done for the analysis of composed security protocols, which, in some cases, can be split into the simpler problem of the analysis of the individual component protocols [55]).

As we mentioned in the introduction, security ceremony analysis is a discipline that is still in its childhood. Our mutation-based approach is, in our opinion, very promising and can investigate scenarios that were not possible before, but there are still a number of open problems, including the more philosophical question if mutations are the optimal way to go. They are possibly not, but they do help model the behavior of human users, and we are currently working with colleagues expert in psychology to investigate if their behavioral studies can be of help. After all, behavioral mutations are investigated commonly in psychology studies (as well as in other disciplines such as epigenetics).

To tackle security ceremonies in full there is thus much more work that needs to be carried out. Other interesting extensions of this work, and of security ceremony analysis in general, are the ones that consider additional abilities of the attacker (e.g., as in [56]) or that consider explicitly the content of messages instead of handling them symbolically as we did here (similar to the case of cyber-physical systems, where message integrity is one of the crucial security properties [57]).

Finally, we plan to consider other, even more complex, case studies, ideally starting from real ceremonies' models or implementations.

References

[1] C.M. Ellison, Ceremony Design and Analysis, *IACR Cryptology ePrint Archive* **399** (2007), 1–17. http://eprint.iacr.org/2007/399.

[2] 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.E. 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: *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, Vol. 7214, Springer, 2012, pp. 267–282. doi:10.1007/978-3-642-28756-5_19.

[3] B. Blanchet, An Efficient Cryptographic Protocol Verifier Based on Prolog Rules, in: *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, IEEE, 2001, pp. 82–96. doi:10.1109/CSFW.2001.930138.

[4] S. Escobar, C. Meadows and J. Meseguer, *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties*, in: *Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures*, Springer, 2009, pp. 1–50. doi:10.1007/978-3-642-03829-7_1.

[5] S. Meier, B. Schmidt, C. Cremers and D.A. Basin, The TAMARIN Prover for the Symbolic Analysis of Security Protocols, in: *Proceedings of the 25th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 8044, Springer, 2013, pp. 696–701. doi:10.1007/978-3-642-39799-8_48.

[6] L.C. Paulson, The Inductive Approach to Verifying Cryptographic Protocols, *Journal of computer security* **6**(1–2) (1998), 85–128. doi:10.3233/JCS-1998-61-205.

[7] D.A. Basin, S. Radomirovic and M. Schläpfer, A Complete Characterization of Secure Human-Server Communication, in: *Proceedings of the IEEE 28th Computer Security Foundations Symposium*, IEEE, 2015, pp. 199–213. doi:10.1109/CSF.2015.21.

[8] D.A. Basin, S. Radomirovic and L. Schmid, Modeling Human Errors in Security Protocols, in: *Proceedings of the 29th IEEE Computer Security Foundations Symposium (CSF)*, IEEE, 2016, pp. 325–340. doi:10.1109/CSF.2016.30.

[9] G. Bella and L. Coles-Kemp, Layered Analysis of Security Ceremonies, in: *Proceedings of the 27th IFIP TC 11 Information Security and Privacy Conference*, IFIP Advances in Information and Communication Technology, Vol. 376, Springer, 2012, pp. 273–286. doi:10.1007/978-3-642-30436-1_23.

[10] M.C. Carlos, J.E. Martina, G. Price and R.F. Custódio, A Proposed Framework for Analysing Security Ceremonies, in: *Proceedings of the International Conference on Security and Cryptography - Volume 1: SECRYPT, (ICETE 2012)*, Scitepress Digital Library, 2012, pp. 440–445, INSTICC. doi:10.5220/0004129704400445.

[11] P. Curzon, R. Rukšėnas and A. Blandford, An approach to formal verification of human-computer interaction, *Formal Aspects of Computing* **19**(4) (2007), 513–550. doi:10.1007/s00165-007-0035-6.

[12] C. Johansen and A. Jøsang, Probabilistic Modelling of Humans in Security Ceremonies, in: *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, Lecture Notes in Computer Science, Vol. 8872, Springer, 2015, pp. 277–292. doi:10.1007/978-3-319-17016-9_18.

[13] J.E. Martina, E. Santos, M.C. Carlos, G. Price and R.F. Custódio, An Adaptive Threat Model for Security Ceremonies, *International Journal of Information Security* **14**(2) (2015), 103–121. doi:10.1007/s10207-014-0253-x.

[14] K. Radke, C. Boyd, J.M.G. Nieto and M. Brereton, Ceremony Analysis: Strengths and Weaknesses, in: *Proceedings of the 26th IFIP International Information Security Conference*, IFIP Advances in Information and Communication Technology, Vol. 354, Springer, 2011, pp. 104–115. doi:10.1007/978-3-642-21424-0_9.

[15] K. Radke and C. Boyd, Security Proofs for Protocols Involving Humans, *The Computer Journal* **60**(4) (2017), 527–540. doi:10.1093/comjnl/bxw066.

[16] L. Viganò, Formal Methods for Socio-technical Security: (Formal and Automated Analysis of Security Ceremonies), in: *Proceedings of the Coordination Models and Languages — 24th IFIP WG 6.1 International Conference, COORDINATION 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022*, LNCS, Vol. 13271, Springer, 2022, pp. 3–14. doi:10.1007/978-3-031-08143-9_1.

[17] D. Sempreboni and L. Viganò, X-Men: A Mutation-Based Approach for the Formal Analysis of Security Ceremonies, in: *Proceedings of the 5th IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2020, pp. 87–104. doi:10.1109/EuroSP48549.2020.00014.

[18] A. Armando, R. Carbone, L. Compagna, J. Cuellar and L. Tobarra, Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-on for Google Apps, in: *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering*, ACM, 2008, pp. 1–10. doi:10.1145/1456396.1456397.

[19] 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).

[20] Y. Jia and M. Harman, An Analysis and Survey of the Development of Mutation Testing, *IEEE Transactions on Software Engineering* **37**(5) (2011), 649–678. doi:10.1109/TSE.2010.62.

[21] M. Büchler, J. Oudinet and A. Pretschner, Security Mutants for Property-Based Testing, in: *Tests and Proofs*, Lecture Notes in Computer Science, Vol. 6706, Springer, 2011, pp. 69–77. doi:10.1007/978-3-642-21768-5_6.

[22] 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) (2015), 684–711. doi:10.1002/stvr.1531.

[23] 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). doi:10.1002/stvr.1685.

[24] L. Viganò, The SPaCIoS Project: Secure Provision and Consumption in the Internet of Services, in: *Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation*, IEEE, 2013, pp. 497–498. doi:10.1109/ICST.2013.75.

[25] X-Men: A Mutation-Based Approach for the Formal Analysis of Security Ceremonies, https://diegosempreboni.github.io/X-Men/, 2021.

[26] TfL (Transport for London), TfL Transparency Strategy, https://tfl.gov.uk/corporate/publications-and-reports/oyster-card.

[27] F.D. Garcia, G. Koning Gans, R. Muijrers, P. Rossum, R. Verdult, R.W. Schreur and B. Jacobs, Dismantling MIFARE Classic, in: *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, Lecture Notes in Computer Science, Springer, 2008, pp. 97–114. doi:10.1007/978-3-540-88313-5_7.

[28] G. de Koning Gans, J.-H. Hoepman and F.D. Garcia, A practical attack on the MiFare Classic, in: *International Conference on Smart Card Research and Advanced Applications*, Lecture Notes in Computer Science, Vol. 5189, Springer, 2008, pp. 267–282. doi:10.1007/978-3-540-85893-5_20.

[29] 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* (2008), 166.

[30] TfL (Transport for London), Incomplete Journeys, https://tfl.gov.uk/fares-and-payments/oyster/using-oyster/incomplete-journeys.

[31] TfL (Transport for London), Card Clash, https://tfl.gov.uk/fares-and-payments/oyster/using-oyster/card-clash.

[32] T. Fábrega, F. Javier, J.C. Herzog and J.D. Guttman, Strand spaces: Proving security protocols correct, *Journal of computer security* **7**(2–3) (1999), 191–230. doi:10.1109/SECPRI.1998.674832.

[33] O. Almousa, S. Mödersheim and L. Viganò, Alice and Bob: Reconciling Formal Models and Implementation, in: *Programming Languages with Applications to Biology and Security — Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday*, Lecture Notes in Computer Science, Vol. 9465, Springer, 2015, pp. 66–85. doi:10.1007/978-3-319-25527-9_7.

[34] M. Abadi and C. Fournet, Mobile Values, New Names, and Secure Communication, in: *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Vol. 36, ACM, 2001, pp. 104–115. doi:10.1145/360204.360213.

[35] D. Dolev and A.C. Yao, On the Security of Public Key Protocols, *IEEE Transactions on information theory* **29**(2) (1983), 198—208. doi:10.1109/TIT.1983.1056650.

[36] The Tamarin User Manual, 2020, https://tamarin-prover.github.io/manual/tex/tamarin-manual.pdf.

[37] S. Mödersheim and L. Viganò, Secure Pseudonymous Channels, in: *Proceedings of the 14th European Conference on Research in Computer Security*, Lecture Notes in Computer Science, Vol. 5789, Springer, 2009, pp. 337–354. doi:10.1007/978-3-642-04444-1_21.

[38] 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 2007/2008/2009 Tutorial Lectures*, Lecture Notes in Computer Science, Vol. 5705, Springer, 2009, pp. 166–194. doi:10.1007/978-3-642-03829-7_6.

[39] B. Schmidt, S. Meier, C. Cremers and D.A. Basin, Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties, in: *Proceedings of the 25th IEEE Computer Security Foundations Symposium*, IEEE, 2012, pp. 78–94. doi:10.1109/CSF.2012.25.

[40] T. Parr, *The Definitive ANTLR 4 Reference*, Pragmatic Bookshelf, 2013.

[41] P.-C. Héam, F. Dadeau, R. Kheddam, G. Maatoug and M. Rusinowitch, A Model-Based Testing Approach for Security Protocols, in: *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, IEEE, 2016, pp. 553–556. doi:10.1109/CSE-EUC-DCABES.2016.240.

[42] L. Viganò and D. Magazzeni, Explainable Security, in: *IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020*, IEEE, 2020, pp. 293–300, A preliminary version appeared as [58]. doi:10.1109/EuroSPW51379.2020.00045.

[43] L. Viganò, Explaining Cybersecurity with Films and the Arts, in: *Imagine Math 7*, M. Emmer and M. Abate, eds, Springer, Cham, 2020, pp. 297–309. doi:10.1007/978-3-030-42653-8_18.

[44] L. Viganò, Don't Tell Me the Cybersecurity Moon is Shining... (Cybersecurity Show and Tell), in: *Imagine Math 8*, M. Emmer, ed., Springer, Cham, 2022, pp. 455–477. doi:10.1007/978-3-030-92690-8_30.

[45] L. Viganò, Nicolas Cage is the Center of the Cybersecurity Universe, in: *Human-Computer Interaction – INTERACT 2021 – 18th IFIP TC 13 International Conference, Proceedings, Part I*, LNCS 12932, Springer, 2021, pp. 14–33. doi:10.1007/978-3-030-85623-6_3.

[46] G. Bella and L. Viganò, Security is Beautiful, in: *Proceedings of the 23rd International Workshop on Security Protocols (SPW'16)*, Lecture Notes in Computer Science, Vol. 9379, Springer, 2015, pp. 247–250. doi:10.1007/978-3-319-26096-9_25.

[47] G. Bella, K. Renaud, D. Sempreboni and L. Viganò, An Investigation into the "Beautification" of Security Ceremonies, in: *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications, ICETE - Volume 2: SECRYPT*, Scitepress Digital Library, 2019, pp. 125–136. doi:10.5220/0007921501250136.

[48] G. Bella, J. Ophoff, R. Karen, D. Sempreboni and L. Viganò, Perceptions of Beauty in Security Ceremonies, *Philosophy & Technology* **35** **(72)** (2022). doi:10.1007/s13347-022-00552-0.

[49] J.E. Martina, T.C.S. de Souza and R.F. Custodio, Ceremonies Formal Analysis in PKI's Context, in: *Proceedings of the International Conference on Computational Science and Engineering - Volume 03*, IEEE, 2009, pp. 392–398. doi:10.1109/CSE.2009.324.

[50] G. Bella, R. Giustolisi and C. Schürmann, Socio-technical formal analysis of TLS certificate validation in modern browsers, *Journal of Computer Security* **30**(3) (2022), 411–433. doi:10.1109/PST.2013.6596067.

[51] D. Sempreboni, G. Bella, R. Giustolisi and L. Viganò, What Are the Threats? (Charting the Threat Models of Security Ceremonies), in: *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications, ICETE - Volume 2: SECRYPT*, Scitepress Digital Library, 2019, pp. 161–172. doi:10.5220/0007924901610172.

[52] B. Beckert and G. Beuster, A Method for Formalizing, Analyzing, and Verifying Secure User Interfaces, in: *Proceedings of the 8th International Conference on Formal Methods and Software Engineering*, Lecture Notes in Computer Science, Vol. 4260, Springer, 2006, pp. 55–73. doi:10.1007/11901433_4.

[53] D. Pavlovic and C. Meadows, Actor-Network Procedures (Extended Abstract), in: *Proceedings of the 8th International Conference on Distributed Computing and Internet Technology (ICDCIT'12)*, Lecture Notes in Computer Science, Vol. 7154, Springer, 2012, pp. 7–26. doi:10.1007/978-3-642-28073-3_2.

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

[55] S. Mödersheim and L. Viganò, Sufficient Conditions for Vertical Composition of Security Protocols, in: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS)*, ACM, 2014, pp. 435–446. doi:10.1145/2590296.2590330.

[56] 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: *Proceedings of the 2nd IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2017, pp. 76–91. doi:10.1109/EuroSP.2017.12.

[57] R. Lanotte, M. Merro, A. Munteanu and L. Viganò, A Formal Approach to Physics-based Attacks in Cyber-Physical Systems, *ACM Trans. Priv. Secur.* **23**(1) (2020), 1–41. doi:10.1145/3373270.

[58] L. Viganò and D. Magazzeni, Explainable Security, *CoRR* (2018), http://arxiv.org/abs/1807.04178.

$$[] \xrightarrow{\mathit{Start}(\mathit{Client},\langle \mathit{SP}\rangle)} [\mathsf{AgSt}(\mathit{Client}, 1, \langle \mathit{SP}\rangle)] \tag{C_0}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Client}, 1, \langle \mathit{SP}\rangle)] \xrightarrow{\mathit{Snd}(\mathit{Client},\mathit{ins},\mathit{SP},\langle \mathit{aenc}(\langle \text{‘}\mathit{get}\text{’},\mathit{URI},\mathit{Client},\text{‘}\mathit{nil}\text{’}\rangle,\mathit{pkSP})\rangle)}\\ &\quad [\mathsf{AgSt}(\mathit{Client}, 2, \langle \mathit{SP}, \mathit{URI}\rangle),\ \mathsf{Out}_{ins}(\mathit{Client}, \mathit{SP}, \langle \mathit{aenc}(\langle \text{‘}\mathit{get}\text{’}, \mathit{URI}, \mathit{Client}, \text{‘}\mathit{nil}\text{’}\rangle, \mathit{pkSP})\rangle)]\end{aligned} \tag{C_1}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Client}, 2, \langle \mathit{SP}, \mathit{URI}\rangle),\ \mathsf{In}_{ins}(\mathit{SP}, \mathit{Client}, \langle \mathit{sign}(\langle \text{‘}\mathit{code302}\text{’}, \mathit{IdP}, \langle \mathit{id}_{SP}, \mathit{SP}\rangle, \mathit{URI}, \text{‘}\mathit{nil}\text{’}\rangle, \mathit{ltkSP})\rangle)]\\ &\quad \xrightarrow{\mathit{Rcv}(\mathit{Client},\mathit{ins},\mathit{SP},\langle \mathit{sign}(\langle \text{‘}\mathit{code302}\text{’},\mathit{IdP},\langle \mathit{id}_{SP},\mathit{SP}\rangle,\mathit{URI},\text{‘}\mathit{nil}\text{’}\rangle,\mathit{ltkSP})\rangle)}\\ &\qquad [\mathsf{AgSt}(\mathit{Client}, 3, \langle \mathit{SP}, \mathit{URI}, \mathit{IdP}, \mathit{id}_{SP}\rangle)]\end{aligned} \tag{C_2}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Client}, 3, \langle \mathit{SP}, \mathit{URI}, \mathit{IdP}, \mathit{id}_{SP}\rangle)] \xrightarrow{\mathit{Snd}(\mathit{Client},\mathrm{sec},\mathit{IdP},\langle \mathit{aenc}(\langle \text{‘}\mathit{get}\text{’},\mathit{IdP},\langle \mathit{id}_{SP},\mathit{SP}\rangle,\mathit{URI},\text{‘}\mathit{nil}\text{’}\rangle,\mathit{pkIdP})\rangle)}\\ &\quad [\mathsf{AgSt}(\mathit{Client}, 4, \langle \mathit{SP}, \mathit{URI}, \mathit{IdP}, \mathit{id}_{SP}\rangle),\\ &\quad \mathsf{Out}_{sec}(\mathit{Client}, \mathit{IdP}, \langle \mathit{aenc}(\langle \text{‘}\mathit{get}\text{’}, \mathit{IdP}, \langle \mathit{id}_{SP}, \mathit{SP}\rangle, \mathit{URI}, \text{‘}\mathit{nil}\text{’}\rangle, \mathit{pkIdP})\rangle)]\end{aligned} \tag{C_3}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Client}, 4, \langle \mathit{SP}, \mathit{URI}, \mathit{IdP}, \mathit{id}_{SP}\rangle),\\ &\ \mathsf{In}_{sec}(\mathit{IdP}, \mathit{Client}, \langle \mathit{aenc}(\langle \mathit{sign}(\langle \text{‘}\mathit{code200}\text{’}, \text{‘}\mathit{nil}\text{’}, \mathit{SP},\\ &\qquad \mathit{sign}(\langle \mathit{id}_{SP}, \mathit{Client}, \mathit{IdP}, \mathit{SP}\rangle, \mathit{ltkIdP}), \mathit{URI}\rangle, \mathit{ltkIdP})\rangle), \mathit{pkClient})\rangle)]\\ &\quad \xrightarrow[\mathit{sign}(\langle \mathit{id}_{SP},\mathit{Client},\mathit{IdP},\mathit{SP}\rangle,\mathit{ltkIdP}),\mathit{URI}\rangle,\mathit{ltkIdP})\rangle),\mathit{pkClient})\rangle)]{\mathit{Rcv}(\mathit{Client},\mathrm{sec},\mathit{IdP},\langle \mathit{aenc}(\langle \mathit{sign}(\langle \text{‘}\mathit{code200}\text{’},\text{‘}\mathit{nil}\text{’},\mathit{SP},}\\ &\qquad [\mathsf{AgSt}(\mathit{Client}, 5, \langle \mathit{SP}, \mathit{URI}, \mathit{IdP}, \mathit{id}_{SP}\rangle)]\end{aligned} \tag{C_4}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Client}, 5, \langle \mathit{SP}, \mathit{URI}, \mathit{IdP}, \mathit{id}_{SP}\rangle)]\\ &\quad \xrightarrow{\mathit{Snd}(\mathit{Client},\mathit{ins},\mathit{SP},\langle \mathit{aenc}(\langle \text{‘}\mathit{post}\text{’},\mathit{SP},\text{‘}\mathit{nil}\text{’},\mathit{sign}(\langle \mathit{id}_{SP},\mathit{Client},\mathit{IdP},\mathit{SP}\rangle,\mathit{ltkIdP}),\mathit{URI}\rangle,\mathit{pkSP})\rangle)}\\ &\qquad [\mathsf{AgSt}(\mathit{Client}, 6, \langle \mathit{SP}, \mathit{URI}, \mathit{IdP}, \mathit{id}_{SP}\rangle),\\ &\qquad \mathsf{Out}_{ins}(\mathit{Client}, \mathit{SP}, \langle \mathit{aenc}(\langle \text{‘}\mathit{post}\text{’}, \mathit{SP}, \text{‘}\mathit{nil}\text{’}, \mathit{sign}(\langle \mathit{id}_{SP}, \mathit{Client}, \mathit{IdP}, \mathit{SP}\rangle, \mathit{ltkIdP}), \mathit{URI}\rangle, \mathit{pkSP})\rangle)]\end{aligned} \tag{C_5}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Client}, 6, \langle \mathit{SP}, \mathit{URI}, \mathit{IdP}, \mathit{id}_{SP}\rangle), \mathsf{In}_{ins}(\mathit{SP}, \mathit{Client}, \langle \mathit{sign}(\langle \text{‘}\mathit{code200}\text{’}, \mathit{URI}, \text{‘}\mathit{resource}\text{’}\rangle, \mathit{ltkSP})\rangle)]\\ &\quad \xrightarrow{\mathit{Rcv}(\mathit{Client},\mathit{ins},\mathit{SP},\langle \mathit{sign}(\langle \text{‘}\mathit{code200}\text{’},\mathit{URI},\text{‘}\mathit{resource}\text{’}\rangle,\mathit{ltkSP})\rangle)} []\end{aligned} \tag{C_6}$$

Figure 25. Agent rules for the *Client* in the SSO ceremony

Appendix A. Agent rules for the SSO ceremony and the Coach Service ceremony

A.1. Agent rules for the SSO ceremony

The agent rules for the *Client*, *IdP* and *SP* in the SSO ceremony are in Figure 25, Figure 26 and Figure 27, respectively.

$$[] \xrightarrow{Start(IdP, \langle \text{'}init\text{'} \rangle)} [\mathsf{AgSt}(IdP, 1, \langle \text{'}init\text{'} \rangle)] \qquad (IdP_0)$$

$$\begin{aligned}&[\mathsf{AgSt}(IdP, 1, \langle \text{'}init\text{'} \rangle),\ \mathsf{In}_{sec}(Client, IdP, \langle aenc(\langle \text{'}get\text{'}, IdP, \langle id_{SP}, SP \rangle, URI, \text{'}nil\text{'} \rangle, pkIdP) \rangle)]\\ &\quad \xrightarrow{Rcv(IdP, \mathrm{sec}, Client, \langle aenc(\langle \text{'}get\text{'}, IdP, \langle id_{SP}, SP \rangle, URI, \text{'}nil\text{'} \rangle, pkIdP) \rangle)}\\ &\qquad [\mathsf{AgSt}(IdP, 2, \langle \text{'}init\text{'} \rangle)] \qquad (IdP_1)\end{aligned}$$

$$\begin{aligned}&[\mathsf{AgSt}(IdP, 2, \langle \text{'}init\text{'} \rangle)]\\ &\quad \xrightarrow[\substack{sign(\langle id_{SP}, Client, IdP, SP \rangle, ltkIdP), URI \rangle, ltkIdP) \rangle), pkClient) \rangle)}]{Snd(IdP, \mathrm{sec}, Client, \langle aenc(\langle sign(\langle \text{'}code200\text{'}, \text{'}nil\text{'}, SP,}\\ &\qquad [\mathsf{AgSt}(IdP, 3, \langle \text{'}init\text{'} \rangle),\\ &\qquad \mathsf{Out}_{sec}(IdP, Client, \langle aenc(\langle sign(\langle \text{'}code200\text{'}, \text{'}nil\text{'}, SP,\\ &\qquad\qquad sign(\langle id_{SP}, Client, IdP, SP \rangle, ltkIdP), URI \rangle, ltkIdP) \rangle), pkClient) \rangle)] \qquad (IdP_2)\end{aligned}$$

Figure 26. Agent rules for the *IdP* in the SSO ceremony

A.2. Agent rules for the Coach Service ceremony

The agent rules for the *Customer*, *WebServer* and *Driver* in the Coach Service are in Figure 28, Figure 29 and Figure 30, respectively.

$$[] \xrightarrow{Start(SP,\langle id_{SP}\rangle)} [\mathsf{AgSt}(SP,1,\langle id_{SP}\rangle)] \tag{SP_0}$$

$$\begin{aligned}&[\mathsf{AgSt}(SP,1,\langle id_{SP}\rangle),\ \mathsf{In}_{ins}(Client,SP,\langle aenc(\langle \text{‘}get\text{’},URI,Client,\text{‘}nil\text{’}\rangle,pkSP)\rangle)]\\ &\quad\xrightarrow{Rcv(SP,ins,Client,\langle aenc(\langle \text{‘}get\text{’},URI,Client,\text{‘}nil\text{’}\rangle,pkSP)\rangle)}\\ &\qquad[\mathsf{AgSt}(SP,2,\langle id_{SP},URI,Client\rangle)]\end{aligned} \tag{SP_1}$$

$$\begin{aligned}&[\mathsf{AgSt}(SP,2,\langle id_{SP},URI,Client\rangle)]\\ &\quad\xrightarrow{Snd(SP,\mathrm{sec},Client,\langle sign(\langle \text{‘}code302\text{’},IdP,\langle id_{SP},SP\rangle,URI,\text{‘}nil\text{’}\rangle,ltkSP)\rangle)}\\ &\qquad[\mathsf{AgSt}(SP,3,\langle id_{SP},URI,Client\rangle),\\ &\qquad\ \mathsf{Out}_{sec}(SP,Client,\langle sign(\langle \text{‘}code302\text{’},IdP,\langle id_{SP},SP\rangle,URI,\text{‘}nil\text{’}\rangle,ltkSP)\rangle)]\end{aligned} \tag{SP_2}$$

$$\begin{aligned}&[\mathsf{AgSt}(SP,3,\langle id_{SP},URI,Client\rangle),\\ &\ \mathsf{In}_{ins}(Client,SP,\langle aenc(\langle \text{‘}post\text{’},SP,\text{‘}nil\text{’},sign(\langle id_{SP},Client,IdP,SP\rangle,ltkIdP),URI\rangle,pkSP)\rangle)]\\ &\quad\xrightarrow{Rcv(SP,ins,Client,\langle aenc(\langle \text{‘}post\text{’},SP,\text{‘}nil\text{’},sign(\langle id_{SP},Client,IdP,SP\rangle,ltkIdP),URI\rangle,pkSP)\rangle)}\\ &\qquad[\mathsf{AgSt}(SP,4,\langle id_{SP},URI,Client\rangle)]\end{aligned} \tag{SP_3}$$

$$\begin{aligned}&[\mathsf{AgSt}(SP,4,\langle id_{SP},URI,Client\rangle)]\\ &\quad\xrightarrow{Snd(SP,ins,Client,\langle sign(\langle \text{‘}code200\text{’},URI,\text{‘}resource\text{’}\rangle,ltkSP)\rangle)}\\ &\qquad[\mathsf{Out}_{ins}(SP,Client,\langle sign(\langle \text{‘}code200\text{’},URI,\text{‘}resource\text{’}\rangle,ltkSP)\rangle)]\end{aligned} \tag{SP_4}$$

Figure 27. Agent rules for the *SP* in the SSO ceremony

A.3. Algorithms for mutations and matching mutations

In this appendix, we give the algorithms for mutations and matching mutations that we did not include in the body of the paper.

Note that the matching algorithms for $\mu^{m(\mu^H_{skip(SR)})}$, $\mu^{m(\mu^H_{skip(RS)})}$ and $\mu^{m(\mu^H_{skip(RSR)})}$ (Algorithms 9, 13 and 15, respectively) are the same as the matching Algorithm 2 for $\mu^{m(\mu^H_{skip(S)})}$; this is because, ultimately, agent A_2 does not receive message m_2 and reacts accordingly. Still, we give all of them as they refer to their corresponding mutation algorithm.

Note also that for the mutations $\mu^H_{skip(SR)}$ and $\mu^H_{skip(RSR)}$ we could remove $Snd(A_2,l_p,A_s,m_p)$ in the mutation of the transition $next(i)$ since H will not receive the message, but we decided to keep it to minimize the changes to agent A_2 (and to keep the uniformity with the other matching algorithms).

Algorithm 8 $\mu^H_{skip(SR)}$: skip $Snd(H,l_2,A_2,m_2)$ in transition i and $Rcv(H,l_3,A_3,m_3)$ in landing transition j as in the trace (4)

```
1: Mutate transition $i$ to
```

$$[] \xrightarrow{\mathit{Start}(\mathit{Customer},\langle \mathit{WebServer},\mathit{Driver}\rangle)} [\mathsf{AgSt}(\mathit{Customer},1,\langle \mathit{WebServer},\mathit{Driver}\rangle)] \qquad (H_0)$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Customer},1,\langle \mathit{WebServer},\mathit{Driver}\rangle)]\\ &\xrightarrow{\mathit{Snd}(\mathit{Customer},\mathrm{sec},\mathit{WebServer},\langle \mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle)}\\ &\quad[\mathsf{AgSt}(\mathit{Customer},2,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle),\\ &\quad\mathsf{Out}_{\mathit{sec}}(\mathit{Customer},\mathit{WebServer},\langle \mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle)] \qquad (H_1)\end{aligned}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Customer},2,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle),\\ &\mathsf{In}_{\mathit{sec}}(\mathit{WebServer},\mathit{Customer},\langle \mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price}\rangle)]\\ &\xrightarrow{\mathit{Rcv}(\mathit{Customer},\mathrm{sec},\mathit{WebServer},\langle \mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price}\rangle)}\\ &\quad[\mathsf{AgSt}(\mathit{Customer},3,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price}\rangle)] \qquad (H_2)\end{aligned}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Customer},3,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price}\rangle)]\\ &\xrightarrow{\mathit{Snd}(\mathit{Customer},\mathrm{sec},\mathit{WebServer},\langle \mathit{price},\text{`}\mathit{ok}\text{'}\rangle)}\\ &\quad[\mathsf{AgSt}(\mathit{Customer},4,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price}\rangle),\\ &\quad\mathsf{Out}_{\mathit{sec}}(\mathit{Customer},\mathit{WebServer},\langle \mathit{price},\text{`}\mathit{ok}\text{'}\rangle)] \qquad (H_3)\end{aligned}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Customer},4,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price}\rangle),\\ &\mathsf{In}_{\mathit{sec}}(\mathit{WebServer},\mathit{Customer},\langle\langle \mathit{Customer},\mathit{tknumber},\mathit{price},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle,\\ &\quad\mathit{senc}(\langle \mathit{Customer},\mathit{tknumber},\mathit{price},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle,\mathit{kS})\rangle)]\\ &\xrightarrow[\mathit{senc}(\langle \mathit{Customer},\mathit{tknumber},\mathit{price},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle,\mathit{kS})\rangle)]{\mathit{Rcv}(\mathit{Customer},\mathrm{sec},\mathit{WebServer},\langle\langle \mathit{Customer},\mathit{tknumber},\mathit{price},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle,}\\ &\quad[\mathsf{AgSt}(\mathit{Customer},5,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price},\mathit{tknumber}\rangle)] \qquad (H_4)\end{aligned}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Customer},5,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price},\mathit{tknumber}\rangle)]\\ &\xrightarrow[\mathit{senc}(\langle \mathit{Customer},\mathit{tknumber},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle,\mathit{kS})\rangle)]{\mathit{Snd}(\mathit{Customer},\mathrm{sec},\mathit{Driver},\langle\langle \mathit{Customer},\mathit{tknumber},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle,}\\ &\quad[\mathsf{AgSt}(\mathit{Customer},6,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price},\mathit{tknumber}\rangle),\\ &\quad\mathsf{Out}_{\mathit{sec}}(\mathit{Customer},\mathit{Driver},\langle\langle \mathit{Customer},\mathit{tknumber},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle,\\ &\qquad\mathit{senc}(\langle \mathit{Customer},\mathit{tknumber},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to}\rangle,\mathit{kS})\rangle)] \qquad (H_5)\end{aligned}$$

$$\begin{aligned}&[\mathsf{AgSt}(\mathit{Customer},6,\langle \mathit{WebServer},\mathit{Driver},\mathit{date},\mathit{dtime},\mathit{from},\mathit{to},\mathit{price},\mathit{tknumber}\rangle),\\ &\mathsf{In}_{\mathit{sec}}(\mathit{Driver},\mathit{Customer},\langle \mathit{Customer},\mathit{tknumber},\mathit{date},\text{`}\mathit{ack}\text{'},\text{`}\mathit{valid}\text{'}\rangle)]\\ &\xrightarrow{\mathit{Rcv}(\mathit{Customer},\mathrm{sec},\mathit{Driver},\langle \mathit{tknumber},\mathit{date},\text{`}\mathit{ack}\text{'},\text{`}\mathit{valid}\text{'}\rangle),\mathit{End}(\mathit{Customer},\text{`}\mathit{ack}\text{'},\text{`}\mathit{valid}\text{'},\mathit{tknumber},\mathit{date})} [] \qquad (H_6)\end{aligned}$$

Figure 28. Agent Rules for the *Customer* in the Coach Service ceremony

$$[\,] \xrightarrow{Start(WebServer,\langle kS,Driver\rangle)} [\mathsf{AgSt}(WebServer, 1, \langle kS, Driver\rangle)] \tag{WS_0}$$

$$\begin{aligned}&[\mathsf{AgSt}(WebServer, 1, \langle kS, Driver\rangle),\ \mathsf{In}_{sec}(Customer, WebServer, \langle date, dtime, from, to\rangle)]\\ &\xrightarrow{Rcv(WebServer,\mathrm{sec},Customer,\langle date,dtime,from,to\rangle)}\\ &\qquad [\mathsf{AgSt}(WebServer, 2, \langle kS, Driver, date, dtime, from, to\rangle]\end{aligned} \tag{WS_1}$$

$$\begin{aligned}&[\mathsf{AgSt}(WebServer, 2, \langle kS, Driver, date, dtime, from, to\rangle]\\ &\xrightarrow{Snd(WebServer,\mathrm{sec},Customer,\langle date,dtime,from,to,price\rangle)}\\ &\qquad [\mathsf{AgSt}(WebServer, 3, \langle kS, Driver, date, dtime, from, to, price\rangle),\\ &\qquad \mathsf{Out}_{sec}(WebServer, Customer, \langle date, dtime, from, to, price\rangle)]\end{aligned} \tag{WS_2}$$

$$\begin{aligned}&[\mathsf{AgSt}(WebServer, 3, \langle kS, Driver, date, dtime, from, to, price\rangle,\ \mathsf{In}_{sec}(Customer, WebServer, \langle price, \text{‘}ok\text{’}\rangle)]\\ &\xrightarrow{Rcv(WebServer,\mathrm{sec},Customer,\langle price,\text{‘}ok\text{’}\rangle)}\\ &\qquad [\mathsf{AgSt}(WebServer, 4, \langle kS, Driver, date, dtime, from, to, price\rangle]\end{aligned} \tag{WS_3}$$

$$\begin{aligned}&[\mathsf{AgSt}(WebServer, 4, \langle kS, Driver, date, dtime, from, to, price\rangle)]\\ &\xrightarrow{\substack{Snd(WebServer,\mathrm{sec},Customer,\langle\langle Customer,tknumber,price,date,dtime,from,to\rangle,\\ senc(\langle Customer,tknumber,price,date,dtime,from,to\rangle,kS)),\\ ValidTicket(WebServer,Customer,\text{‘}tn\text{’},tknumber,date)}}\\ &\qquad [\mathsf{AgSt}(WebServer, 5, \langle kS, Driver, date, dtime, from, to, price, tknumber\rangle,\\ &\qquad \mathsf{Out}_{sec}(WebServer, Customer, \langle\langle Customer, tknumber, price, date, dtime, from, to\rangle,\\ &\qquad\qquad senc(\langle Customer, tknumber, price, date, dtime, from, to\rangle, kS)\rangle)]\end{aligned} \tag{WS_4}$$

Figure 29. Agent Rules for the *WebServer* in the Coach Service ceremony

```
            $\mathsf{AgSt}(H, i, kn_i),\ Pre_i,\ Rcv(H, l_1, A_1, m_1) \rightarrow$
            $\mathsf{AgSt}(H, i+1, kn_{i+1}),\ Post_{i+1}$
2:  $[\![kn_j]\!]^{\mu} ::= kn_j = kn_{i+1}$    ▷ Since $\Sigma_2$, and thus $[\![\Sigma_2]\!]^{\mu}$, does not contain a transition in which $H$
    receives new information
3:  $[\![kn_{j+1}]\!]^{\mu} ::= kn_j \cup Pre_j$
4:  build all $[\![m_4]\!]^{\mu} \in \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^{\mu}$
5:  mutate transition $j$ to
            $\mathsf{AgSt}(H, j, kn_j),\ Pre_j \rightarrow$
            $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^{\mu}),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^{\mu})$
```

$$[] \xrightarrow{Start(Driver,\langle kS\rangle)} [\mathsf{AgSt}(Driver, 1, \langle kS\rangle)] \qquad (D_0)$$

$$\begin{array}{l}
[\mathsf{AgSt}(Driver, 1, \langle kS\rangle), \\
\ \mathsf{In}_{sec}(Customer, Driver, \langle\langle Customer, tknumber, price, date, dtime, from, to\rangle, \\
\qquad senc(\langle Customer_{kS}, tknumber_{kS}, price_{kS}, date_{kS}, dtime_{kS}, from_{kS}, to_{kS}\rangle, kS)\rangle)] \\
\qquad \xrightarrow{\substack{Rcv(Driver,\mathrm{sec},Customer,\langle\langle Customer,tknumber,price,date,dtime,from,to\rangle, \\ senc(\langle Customer_{kS},tknumber_{kS},price_{kS},date_{kS},dtime_{kS},from_{kS},to_{kS}\rangle,kS)}} \\
\qquad\qquad [\mathsf{AgSt}(Driver, 2, \langle kS, Customer, tknumber, date, dtime, from, to\rangle] \qquad (D_1)
\end{array}$$

$$\begin{array}{l}
[\mathsf{AgSt}(Driver, 2, \langle kS, Customer, tknumber, date, dtime, from, to\rangle)] \\
\qquad \xrightarrow{\substack{Snd(Driver,\mathrm{sec},Customer,\langle tknumber,date,\text{‘}ack\text{’},\text{‘}valid\text{’}\rangle) \\ Eq(tknumber,tknumber_{kS}), \\ Eq(Customer,Customer_{kS}), \\ Eq(price,price_{kS}), \\ Eq(date,date_{kS}), \\ Eq(dtime,dtime_{kS}), \\ Eq(from,from_{kS}), \\ Eq(to,to_{kS})}} \\
\qquad\qquad [\mathsf{AgSt}(Driver, 3, \langle kS, Customer, tknumber, date, dtime, from, to\rangle), \\
\qquad\qquad\ \mathsf{Out}_{sec}(Driver, Customer, \langle Customer, tknumber, date, \text{‘}ack\text{’}, \text{‘}valid\text{’}\rangle)] \quad (D_2)
\end{array}$$

Figure 30. Agent Rules for the *Driver* in the Coach Service ceremony

Algorithm 9 Matching mutation $\mu^{m(\mu^{H}_{skip(SR)})}$ for $\mu^{H}_{skip(SR)}$

```
1: Consider the transition $next(i)$ in $\Sigma_2$ that immediately follows the mutated human transition $i$,
   i.e.
       $\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \to$
       $\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$,
   where $A_s$ is one of the other agents and $l_p$ and $m_p$ are some channel and message as specified
   in $\Sigma_2$
2: remove $Rcv(A_2, l_2, H, m_2)$ from $next(i)$
3: $[\![kn_{x+1}]\!]^{\mu} ::= [\![kn_x]\!]^{\mu} \cup Pre_x$, where $[\![kn_x]\!]^{\mu} ::= kn_{x-1}$
4: build all $[\![m_p]\!]^{\mu} \in \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^{\mu}$
5: mutate the transition to
       $\mathsf{AgSt}(A_2, x, [\![kn_x]\!]^{\mu}),\ Pre_x \to$
       $\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^{\mu}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^{\mu})$
```

```
6: let $h ::= next(i)$
7: if the trace contains a transition $next(h)$ of the form
        $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$ then
8:     if this $next(h)$ is actually $H$'s landing transition $j$ already considered in Algorithm 8 then
9:         go to 7 with $h ::= next(h)$
10:    else
11:        replace $m_p$ with $[\![m_p]\!]^\mu$
12:        $[\![kn_{s+1}]\!]^\mu ::= [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$, where $[\![kn_s]\!]^\mu ::= kn_{s-1}$
13:        build all $[\![m_{p+1}]\!]^\mu \in \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated
   by $[\![kn_{s+1}]\!]^\mu$
14:        mutate the transition to
               $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
               $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
15:        go to 7 with $h ::= next(h)$
```

Algorithm 10 $\mu^{H}_{skip(R)}$: skip $Rcv(H, l_1, A_1, m_1)$ in transition i, with landing transition j as in the trace (5)

```
1: $[\![kn_{i+1}]\!]^{\mu} ::= kn_i \cup Pre_i$
2: build all $[\![m_2]\!]^{\mu} \in \{(format(m_2))(m) \mid m \in submsg(m_2)\}$ that can be generated by $[\![kn_{i+1}]\!]^{\mu}$
3: mutate transition $i$ to
           $\mathsf{AgSt}(H, i, kn_i),\ Pre_i \rightarrow$
           $\mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^{\mu}),\ Post_{i+1},\ Snd(H, l_2, A_2, [\![m_2]\!]^{\mu})$
4: $[\![kn_j]\!]^{\mu} ::= [\![kn_{i+1}]\!]^{\mu}$    ▷ Since $\Sigma_2$, and thus $[\![\Sigma_2]\!]^{\mu}$, does not contain a transition in which $H$ receives new information
        ▷ There are two cases, depending on which message is sent by $A_3$, the original $m_3$ or its mutation $[\![m_3]\!]^{\mu}$
5: if $[\![\Sigma_2]\!]^{\mu}$ contains a transition with $Snd(A_3, l_3, H, m_3)$ in its conclusions then
6:     $[\![kn_{j+1}]\!]^{\mu} ::= [\![kn_j]\!]^{\mu} \cup \{m_3\} \cup Pre_j$
7:     build all $[\![m_4]\!]^{\mu} \in \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^{\mu}$
8:     mutate transition $j$ to
           $\mathsf{AgSt}(H, j, [\![kn_j]\!]^{\mu}),\ Pre_j,\ Rcv(H, l_3, A_3, m_3) \rightarrow$
           $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^{\mu}),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^{\mu})$
9: else    ▷ $[\![\Sigma_2]\!]^{\mu}$ contains a transition with $Snd(A_3, l_3, H, [\![m_3]\!]^{\mu})$ in its conclusion for some mutation $[\![m_3]\!]^{\mu}$ defined in Algorithm 11
10:    $[\![kn_{j+1}]\!]^{\mu} ::= [\![kn_j]\!]^{\mu} \cup \{[\![m_3]\!]^{\mu}\} \cup Pre_j$
11:    build all $[\![m_4]\!]^{\mu} \in \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^{\mu}$
12:    mutate transition $j$ to
           $\mathsf{AgSt}(H, j, [\![kn_j]\!]^{\mu}),\ Pre_j,\ Rcv(H, l_3, A_3, [\![m_3]\!]^{\mu}) \rightarrow$
           $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^{\mu}),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^{\mu})$
```

Algorithm 11 Matching mutation $\mu^{m(\mu^{H}_{skip(R)})}$ for $\mu^{H}_{skip(R)}$

```
1: Consider the transition $next(i)$ in $\Sigma_2$ that immediately follows the mutated human transition $i$,
   i.e.
       $\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
       $\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p)$,
   where $A_s$ is one of the other agents and $l_p$ and $m_p$ are some channel and message as specified
   in $\Sigma_2$
2: replace $Rcv(A_2, l_2, H, m_2)$ with $Rcv(A_2, l_2, H, [\![m_2]\!]^\mu)$
3: $[\![kn_{x+1}]\!]^\mu ::= [\![kn_x]\!]^\mu \cup \{[\![m_2]\!]^\mu\} \cup Pre_x$, where $[\![kn_x]\!]^\mu ::= kn_{x-1}$
4: build all $[\![m_p]\!]^\mu \in \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^\mu$
5: mutate the transition to
       $\mathsf{AgSt}(A_2, x, [\![kn_x]\!]^\mu),\ Pre_x,\ Rcv(A_2, l_2, H, [\![m_2]\!]^\mu) \rightarrow$
       $\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^\mu),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^\mu)$
6: let $h ::= next(i)$
7: **if** the trace contains a transition $next(h)$ of the form
       $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
       $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$ **then**
8:     **if** this $next(h)$ is actually $H$'s landing transition $j$ already considered in Algorithm 10 **then**
9:         **go to** 7 **with** $h ::= next(h)$
10:    **else**
11:        replace $m_p$ with $[\![m_p]\!]^\mu$
12:        $[\![kn_{s+1}]\!]^\mu ::= [\![kn_s]\!]^\mu \cup Pre_s \cup [\![m_p]\!]^\mu$, where $[\![kn_s]\!]^\mu ::= kn_{s-1}$
13:        build all $[\![m_{p+1}]\!]^\mu \in \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated
   by $[\![kn_{s+1}]\!]^\mu$
14:        mutate the transition to
               $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^\mu),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^\mu) \rightarrow$
               $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^\mu),\ Post_{s+1}, Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^\mu)$
15:        **go to** 7 **with** $h ::= next(h)$
```

Algorithm 12 $\mu^{H}_{skip(RS)}$: 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 as in the trace (6)

```
1: $[\![kn_{i+1}]\!]^\mu ::= kn_i \cup Pre_i$
2: mutate transition $i$ to
           $\mathsf{AgSt}(H, i, kn_i),\ Pre_i \rightarrow$
           $\mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^\mu),\ Post_{i+1}$
3: $[\![kn_j]\!]^\mu ::= [\![kn_{i+1}]\!]^\mu$    ▷ *Since $\Sigma_2$, and thus $[\![\Sigma_2]\!]^\mu$, does not contain a transition in which $H$ receives new information*
       ▷ *There are two cases, depending on which message is sent by $A_3$, the original $m_3$ or its mutation $[\![m_3]\!]^\mu$*
4: **if** $[\![\Sigma_2]\!]^\mu$ contains a transition with $Snd(A_3, l_3, H, m_3)$ in its conclusions **then**
5:     $[\![kn_{j+1}]\!]^\mu ::= [\![kn_j]\!]^\mu \cup \{m_3\} \cup Pre_j$
6:     build all $[\![m_4]\!]^\mu \in \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^\mu$
7:     mutate transition $j$ to
           $\mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j,\ Rcv(H, l_3, A_3, m_3) \rightarrow$
           $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
8: **else**    ▷ *$[\![\Sigma_2]\!]^\mu$ contains a transition with $Snd(A_3, l_3, H, [\![m_3]\!]^\mu)$ in its conclusion for some mutation $[\![m_3]\!]^\mu$ defined in Algorithm 11*
9:     $[\![kn_{j+1}]\!]^\mu ::= [\![kn_j]\!]^\mu \cup \{[\![m_3]\!]^\mu\} \cup Pre_j$
10:    build all $[\![m_4]\!]^\mu \in \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^\mu$
11:    mutate transition $j$ to
           $\mathsf{AgSt}(H, j, [\![kn_j]\!]^\mu),\ Pre_j,\ Rcv(H, l_3, A_3, [\![m_3]\!]^\mu) \rightarrow$
           $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^\mu),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^\mu)$
```

Algorithm 13 Matching mutation $\mu^{m(\mu^{H}_{skip(RS)})}$ for $\mu^{H}_{skip(RS)}$

```
1: Consider the transition $next(i)$ in $\Sigma_2$ that immediately follows the mutated human transition $i$, i.e.
       $\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \rightarrow$
       $\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p),$
   where $A_s$ is one of the other agents and $l_p$ and $m_p$ are some channel and message as specified in $\Sigma_2$
2: remove $Rcv(A_2, l_2, H, m_2)$ from $next(i)$
3: $[\![kn_{x+1}]\!]^{\mu} ::= [\![kn_x]\!]^{\mu} \cup Pre_x$, where $[\![kn_x]\!]^{\mu} ::= kn_{x-1}$
4: build all $[\![m_p]\!]^{\mu} \in \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^{\mu}$
5: mutate the transition to
       $\mathsf{AgSt}(A_2, x, [\![kn_x]\!]^{\mu}),\ Pre_x\ \rightarrow$
       $\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^{\mu}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^{\mu})$
6: let $h ::= next(i)$
7: **if** the trace contains a transition $next(h)$ of the form
        $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, m_p) \rightarrow$
        $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$ **then**
8:     **if** this $next(h)$ is actually $H$'s landing transition $j$ already considered in Algorithm 12 **then**
9:         **go to** 7 **with** $h ::= next(h)$
10:    **else**
11:        replace $m_p$ with $[\![m_p]\!]^{\mu}$
12:        $[\![kn_{s+1}]\!]^{\mu} ::= [\![kn_s]\!]^{\mu} \cup Pre_s \cup [\![m_p]\!]^{\mu}$, where $[\![kn_s]\!]^{\mu} ::= kn_{s-1}$
13:        build all $[\![m_{p+1}]\!]^{\mu} \in \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated by $[\![kn_{s+1}]\!]^{\mu}$
14:        mutate the transition to
               $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^{\mu}),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^{\mu}) \rightarrow$
               $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^{\mu}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^{\mu})$
15:        **go to** 7 **with** $h ::= next(h)$
```

Algorithm 14 $\mu^{H}_{skip(RSR)}$: skip $Rcv(H, l_1, A_1, m_1)$ and $Snd(H, A_2, l_2, m_2)$ in transition i, and $Rcv(H, l_3, A_3, [\![m_3]\!]^{\mu})$ in landing transition j as in the trace (7)

```
1: $[\![kn_{i+1}]\!]^{\mu} ::= kn_i \cup Pre_i$
2: mutate transition $i$ to
           $\mathsf{AgSt}(H, i, kn_i),\ Pre_i \rightarrow$
           $\mathsf{AgSt}(H, i+1, [\![kn_{i+1}]\!]^{\mu}),\ Post_{i+1}$
3: $[\![kn_j]\!]^{\mu} ::= [\![kn_{i+1}]\!]^{\mu}$      ▷ Since $\Sigma_2$, and thus $[\![\Sigma_2]\!]^{\mu}$, does not contain a transition in which H
   receives new information
4: $[\![kn_{j+1}]\!]^{\mu} ::= [\![kn_j]\!]^{\mu} \cup Pre_j$
5: build all $[\![m_4]\!]^{\mu} \in \{(format(m_4))(m) \mid m \in submsg(m_4)\}$ that can be generated by $[\![kn_{j+1}]\!]^{\mu}$
6: mutate transition $j$ to
           $\mathsf{AgSt}(H, j, [\![kn_j]\!]^{\mu}),\ Pre_j \rightarrow$
           $\mathsf{AgSt}(H, j+1, [\![kn_{j+1}]\!]^{\mu}),\ Post_{j+1},\ Snd(H, l_4, A_4, [\![m_4]\!]^{\mu})$
```

Algorithm 15 Matching mutation $\mu^{m(\mu^{H}_{skip(RSR)})}$ for $\mu^{H}_{skip(RSR)}$

```
1: Consider the transition $next(i)$ in $\Sigma_2$ that immediately follows the mutated human transition $i$,
   i.e.
       $\mathsf{AgSt}(A_2, x, kn_x),\ Pre_x,\ Rcv(A_2, l_2, H, m_2) \to$
       $\mathsf{AgSt}(A_2, x+1, kn_{x+1}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, m_p),$
   where $A_s$ is one of the other agents and $l_p$ and $m_p$ are some channel and message as specified
   in $\Sigma_2$
2: remove $Rcv(A_2, l_2, H, m_2)$ from $next(i)$
3: $[\![kn_{x+1}]\!]^{\mu} ::= [\![kn_x]\!]^{\mu} \cup Pre_x$, where $[\![kn_x]\!]^{\mu} ::= kn_{x-1}$
4: build all $[\![m_p]\!]^{\mu} \in \{(format(m_p))(m) \mid m \in submsg(m_p)\}$ that can be generated by $[\![kn_{x+1}]\!]^{\mu}$
5: mutate the transition to
       $\mathsf{AgSt}(A_2, x, [\![kn_x]\!]^{\mu}),\ Pre_x\ \to$
       $\mathsf{AgSt}(A_2, x+1, [\![kn_{x+1}]\!]^{\mu}),\ Post_{x+1},\ Snd(A_2, l_p, A_s, [\![m_p]\!]^{\mu})$
6: let $h ::= next(i)$
7: **if** the trace contains a transition $next(h)$ of the form
       $\mathsf{AgSt}(A_s, s, kn_s),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, m_p) \to$
       $\mathsf{AgSt}(A_s, s+1, kn_{s+1}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, m_{p+1})$ **then**
8:     **if** this $next(h)$ is actually $H$'s landing transition $j$ already considered in Algorithm 14 **then**
9:         **go to** 7 **with** $h ::= next(h)$
10:    **else**
11:        replace $m_p$ with $[\![m_p]\!]^{\mu}$
12:        $[\![kn_{s+1}]\!]^{\mu} ::= [\![kn_s]\!]^{\mu} \cup Pre_s \cup [\![m_p]\!]^{\mu}$, where $[\![kn_s]\!]^{\mu} ::= kn_{s-1}$
13:        build all $[\![m_{p+1}]\!]^{\mu} \in \{(format(m_{p+1}))(m) \mid m \in submsg(m_{p+1})\}$ that can be generated
   by $[\![kn_{s+1}]\!]^{\mu}$
14:        mutate the transition to
               $\mathsf{AgSt}(A_s, s, [\![kn_s]\!]^{\mu}),\ Pre_s,\ Rcv(A_s, l_p, A_{s-1}, [\![m_p]\!]^{\mu}) \to$
               $\mathsf{AgSt}(A_s, s+1, [\![kn_{s+1}]\!]^{\mu}),\ Post_{s+1},\ Snd(A_s, l_p, A_{s+1}, [\![m_{p+1}]\!]^{\mu})$
15:        **go to** 7 **with** $h ::= next(h)$
```