Honest Users Make Honest Mistakes: A Framework for Analysing eID Protocols

Ole Martin Edstrøm
Norwegian University of Science and Technology
Trondheim, Norway
ole.m.edstrom@ntnu.no

Kristian Gjøsteen
Norwegian University of Science and Technology
Trondheim, Norway
kristian.gjosteen@ntnu.no

Hans Heum
Norwegian University of Science and Technology
Trondheim, Norway
hans.heum@ntnu.no

Sjouke Mauw
University of Luxembourg
Esch-sur-Alzette, Luxembourg
sjouke.mauw@uni.lu

Felix Stutz
University of Luxembourg
Esch-sur-Alzette, Luxembourg
felix.stutz@uni.lu

Abstract

Electronic identification (eID) protocols and federated identity management systems play an increasingly important role in our modern society, both on the internet through services from Google and others, and through the eIDAS regulation in Europe. A key feature of eID protocols is that humans are intimately involved in the protocol, often responsible for critical security steps. Traditional security analyses of such protocols typically assume flawless user behaviour, yet widespread real-world adoption makes user mistakes inevitable.

We present a framework for analysing the security of eID protocols that can model users making mistakes. It is suitable for automated analysis with Tamarin and supports fine-grained corruption modelling of protocol actors. We demonstrate the framework's utility by describing and analysing common eID protocols based on passwords, mobile applications and authentication tokens, as well as by systematically evaluating the impact of various combinations of user mistakes on security.

1 Introduction

Passwords are ubiquitous in our modern life. For most online accounts, we initially sign up with an identifier (a user name or e-mail address) and a password. However, the use of solely the identifier and password easily becomes a security issue. While diligent use of a password manager may help address the most glaring vulnerabilities associated with passwords, better solutions are desirable [3].

In the past decades, there have been two major developments in how people log in to websites and other accounts.

First, there has been a push towards using authentication mechanisms beyond usernames and passwords, such as Multifactor Authentication (MFA) schemes in which authentication relies on a password and a phone for instance. These range from simple text messages sent to the user's mobile phone, via various apps installed on secondary devices, to dedicated authentication tokens (for example using FIDO2[1] for key management). Thus, an attacker obtaining the password or the security device only will still be unable to access the user's account. To enjoy this enhanced security however, a user needs to set up an MFA scheme for every individual account, which can be burdensome, especially if the account should be linked to some real-world identity.

This is where the second development comes into play: with *federated identity management systems* [4], users only need to register their credentials once. Such systems make it easier and more economical to adopt more advanced technologies like MFA, which can provide greater overall security for a system. Basically, they act as *identity provider* and other websites, usually called *relying parties* (which we will simply call *servers* later on), can rely on them for the login mechanism. For instance, with their *electronic identification (eID) protocols*, some countries act as identity provider, handling authentication for sensitive services like bank applications[2]. Several big companies, e.g. Google[3] and Meta[4], offer a similar service for their customers to allow them to use their credentials to log into services. Because of this similarity, we will also use the term eID protocol for the security protocols used by federated identity management systems in this work.

While both governmental services and IT companies can act as identity provider, their guarantees differ: the former actually vouches for the identity of the user, while the latter usually does not check their identity but simply ensures continuity with respect to the user and its identifier; both will likely see increased adoption going forward. Such adoption at scale calls for sophisticated security analyses to protect users, their accounts, and assets.

While protocols like the ones described above have been extensively analysed and undergo a rigorous standardisation process, most existing analyses assume that users behave perfectly according to specification. This means that each user conducts all checks accurately and makes no mistakes when logging via an eID protocol. When such protocols become omnipresent, this assumption is easily falsified, as even honest users are prone to make mistakes [16]. For instance, a second factor (such as a security token) may display the website that is currently being logged in to, in order to protect against an adversary that tries to hijack the login procedure. While a trustworthy display could in principle thwart such

[1] https://fidoalliance.org/fido2/

This work was partially supported by the Norwegian Research Council through the project 'Societal Security and Digital Identities' (project number 320785) and the Luxembourg National Research Fund (FNR) under the grant agreement C22/IS/17238244/AVVA..

[2] E.g. the Estonian ID-card, https://www.id.ee/en/.
[3] https://www.google.com/account/about/sign-in-with-google/
[4] https://developers.secure.facebook.com/docs/facebook-login/overview

an attack, this is predicated on the user actually checking the value being displayed.

Since user mistakes can affect the security of the protocol, it is important that eID protocol interfaces are designed to minimise the likelihood of error, such as an app requiring the user to select a matching value among several options, instead of just checking that two values (displayed across devices) match. Our model captures the positive effect of these and other countermeasures via security notions that allow for honest users that still fail to follow the protocol to specification.

Some eID protocols will be subject to legal regulation [10], and an important question is the residual risk of the technical system which must be accepted by system owners, users, relying parties and regulators. This is of particular importance when the system fails and the failure is caused by user mistakes.

There are various kinds of mistakes a user can make. Usually, it is a combination of failing to perform the required checks, e.g. server identities, identity provider URL and unique information (nonces) used to link sessions. Combining the various kinds of user mistakes leads to a high multiplicity of attack scenarios, making the analysis a monumental task.

Contributions. To fill this gap, we propose a framework for analysing the security of eID protocols, providing

- a general definition of eID protocols;
- a hierarchical set of security notions, matching real-world eID protocols;
- a natural modelling of user mistakes;
- a fine-grained notion of corruptions, allowing for precise trust assumptions;
- automated verification using Tamarin [19].

The last point is important, since already in this work we study 15 distinct eID protocols under a variety of trust assumptions (4 in the main setting, 8 for the fine-grained user behaviour study). Tamarin requires between 2 minutes and 5 hours to analyse each protocol. A summary of our results can be found in Table 1, with a more detailed discussion in Section 3.

Past and future work. Section 4 compares our approach to that of previous work, and Section 5.1 presents some possible future directions.

2 Modelling

eID protocols typically rely on cryptographic protocols such as TLS 1.3 [21] and/or digital signatures to establish secure channels between interacting parties. Dolev-Yao models [6] are suitable for such settings, assuming confidential and (point-to-point) authenticated channels, and allowing analysis to focus on the (user) authentication protocol under scrutiny.

In this section, we provide the formal foundation for our Dolev-Yao model, and explain how we use it to formalise our informal message sequence charts (MSCs) and state machines which describe our eID protocols, and define security notions for eID protocols. We will use the eID protocol given in Fig. 1, based on one way to use Web Authentication (WebAuthn) [15], as a running example. A more detailed discussion of its analysis can be found in Section 3.

The MSC provides an intuitive description of the protocol. Each role is represented by a name-role pair (identifying the *agent* executing the role) and initial register values (the agent's initial knowledge), with arrows from and to the dotted line denoting sent and received messages, and boxes denoting local computations.

We see that a *user* logs into a *server* using a *browser tab*. An *identity provider* vouches for the user by signing the server's challenge, once the user's *token*[5] has signed the identity provider's challenge, which it will only do if the user approves the request with its token; the identity provider communicates with the token via the browser tab. All communication either happens via secure channels through insecure networks (in practice, this would be achieved using TLS; our the Dolev-Yao model gives us secure channels immediately), or via effectively secure physical channels (e.g. cabled communication between computer and token, or the user reading things on the computer's screen).

Our MSC does not fully describe the protocol; for instance, it does not show what a role does if signature verification fails. We use state machines to precisely describe a protocol, and Fig. 1 shows the user role: states are labeled with local computations, transitions are labeled with messages or computation outcomes. Hats denote values taken from registers, while bars denotes registers taking values from received message terms or computation outcomes. Note also that the user receives a nonce `cid`. This is a modelling artefact, to be discussed in detail in Section 2.3.

2.1 Multiset Rewriting Rules

We use Tamarin [19] for automated verification. Tamarin works with multisets of so-called *facts*, which can for instance model agent state, events, and messages. The protocol and adversary are defined in terms of rules specifying how the states evolve, the *multiset rewriting rules*. Our presentation is inspired by Dreier et al. [8], though we omit details that are not relevant to our context.

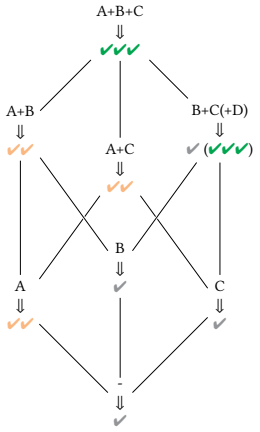
2.1.1 Message algebra and equational theory. Messaging between agents is at the heart of protocols. In our setting, *messages* are terms from an order-sorted term algebra, equipped with an equational theory. Besides modelling communication, messages are also a building block for representing the state and knowledge of agents and the adversary.

We define three sorts for messages: *msg* for all terms, and the two incomparable sub-sorts *fr* (to denote fresh values) and *pub* (to denote public values). For each sort s, we have a countable set of variables $\mathcal{V}_s$ and a countable set of names $\mathcal{N}_s$, with their union over all sorts denoted by $\mathcal{N}$ and $\mathcal{V}$. To construct messages from names, we assume a signature Σ_{Op}, a finite set of function symbols, each of which has a unique arity. We denote by $\mathcal{T}_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$ the set of terms using function symbols from Σ_{Op} over variables from $\mathcal{V}$ and names from $\mathcal{N}$. We use three function symbols for digital signatures: $vk(\cdot)$ makes a verification key from a signing key, $\textsc{Sign}(m, sk)$ signs m with a signing key sk, and $\textsc{Verify}(m', m, vk)$ verifies that m' is a signature on m under verification key vk. In Fig. 1 for example,

[5] A *token* is a small computer that can take on a variety of physical forms. Traditionally, the smart card or USB stick form factor connected to the user's computer via a cable has been prevalent. Modern tokens are often connected via radio (e.g. Bluetooth). Nowadays, secure elements in the user's computer are also common.

Table 1: The table on the left contains a summary of our results, while the diagram on the right shows a fine-grained user behaviour study. The protocols (described in Section 3) achieve no security (✘), recent aliveness (✔), weak security (✔✔), security (✔✔✔) and honesty-preservation (†), all defined in Section 2.3. We consider user behaviour that is perfect (do every check) or imperfect (no checks). In the protocol App Cmp⁺ the user should check the (A) server identities, (B) infrastructure URL and (C) nonces. The diagram shows the security effect of doing only some of these checks. Our main setting is that users interact with both honest and corrupt browser tabs, but we also evaluate protocols when (D) users only interact with honest browser tabs.

User browser tabs	honest & corrupt		all honest	
User behaviour	imperfect	perfect	imperfect	perfect
Password	✘	✘	✘	✔✔✔
App	✔	✔	✔	✔
App⁺	✔	✔✔	✔	✔✔
App Cmp	✔	✔	✔	✔✔✔
App Cmp⁺	✔	✔✔✔	✔	✔✔✔
App Write	✔	✔	✔	✔✔✔
App Write⁺	✔	✔✔✔	✔	✔✔✔
Two-factor	✔	✔	✔	✔†
Two-factor⁺	✔	✔✔	✔	✔✔†
Two-factor Cmp	✔	✔	✔	✔✔✔
Two-factor Cmp⁺	✔	✔✔✔	✔	✔✔✔
Two-factor Write	✔	✔	✔	✔✔✔
Two-factor Write⁺	✔	✔✔✔	✔	✔✔✔
WebAuthn	✔	✔	✔✔✔	✔✔✔
WebAuthn⁺	✔	✔✔✔	✔✔✔	✔✔✔



both the signing and verification function is used by the identity provider.

An *equation* $s = t$ is an unordered pair of terms $s, t \in \mathcal{T}_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N})$. Given a set of equations E, an *equational theory* (Σ_{Op}, E) is the smallest congruence closure containing all instances of E. We denote by $s =_E t$ two terms s and t that are related by any equational theory (Σ_{Op}, E). The set Σ_{Op} is implicit, as it is usually fixed or unambiguous. We use similar notation for other relations and operators modulo the equational theory, e.g. $\in_E$ for (multi)set membership.

The equation $\textsc{Verify}(\textsc{Sign}(x, y), x, vk(y)) = \texttt{True}$ models signature verification as an equational theory. The interpretation is straightforward: for any x and y, if one signs x with y ($\textsc{Sign}(x, y)$), then verification will only return `True` when input the same x and the verification key $vk(y)$ generated from y. In Fig. 1, we assume that verification keys were distributed prior to running the protocol, i.e. $vk(sk_t) = vk_t$. Tamarin provides several built-in equational theories, such as for encryption, but we will not need them because we shall assume secure channels.

2.1.2 Facts, transition systems and modelling protocols. Facts are versatile and can model events, states of agents, general information, messages in flight, and hence protocol states in general. The set of *facts* is

$$\mathcal{F} = \{F(t_1, \ldots, t_n) \mid t_i \in \mathcal{T}_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N}) \text{ and } F \in \Sigma_{Fact} \text{ with arity } n\},$$

where Σ_{Fact} denotes an unsorted signature for facts. We denote the set of multisets of facts by $\mathcal{F}^\#$ and the set of ground facts as $\mathcal{G}^\#$ (facts solely using names from $\mathcal{N}$ and no variables from $\mathcal{V}$). We denote multiset operations by a superscript #, e.g. $\cup^\#$. Facts are either *linear* or *persistent*. The former will be consumed by rules, the latter will not.

Facts model public information, such as what the verification key of the identity provider is. These are typically modelled as persistent facts, while states of agents are linear facts which will be consumed by rules. For instance, we can use the fact $\mathsf{U}_0(u, b, t, s_\mathrm{I})$ to represent a user agent's initial state (and knowledge) while we model a message from browser b to user u as $\mathtt{Msg}(b, u, (5, s'_\mathrm{I}, s))$.

We use *events* that are facts with signatures $\Sigma_{Ev} = \{\mathtt{itt}, \mathtt{blf}\} \subseteq \Sigma_{Fact}$:

$$\mathcal{F}_{Ev} = \{F(t_1, \ldots, t_n) \mid t_i \in \mathcal{T}_{\Sigma_{Op}}(\mathcal{V}, \mathcal{N}) \text{ and } F \in \Sigma_{Ev} \text{ with arity } n\} \subseteq \mathcal{F}.$$

We use the event $\mathtt{itt}(\ldots)$ to model user intent, while the event $\mathtt{blf}(\ldots)$ models server belief. These events will be used to define the security notions.

State changes are modelled by *multiset rewriting rules*, which are 3-tuples of the form (l, a, r), where $l \in \mathcal{F}^\#$ are *premises*, $a \in \mathcal{F}^\#$ are *actions*, and $r \in \mathcal{F}^\#$ are *conclusions*. We sometimes denote this by $l \,-[\, a \,]\!\rightarrow r$. It is convenient to split the premises l into linear facts $\mathrm{lfacts}(l)$ and persistent facts $\mathrm{pfacts}(l)$, i.e. $l = \mathrm{lfacts}(l) \uplus^\# \mathrm{pfacts}(l)$ (where $\uplus^\#$ denotes disjoint union). For what follows, let us assume a set of rules R. We write $\mathrm{ginsts}(R)$ for the set of ground instances of R, i.e. where all terms contain no variables and only names. The *semantics* of R is given by a transition relation $\rightarrow_R \subseteq \mathcal{G}^\# \times \mathcal{G}^\# \times \mathcal{G}^\#$,

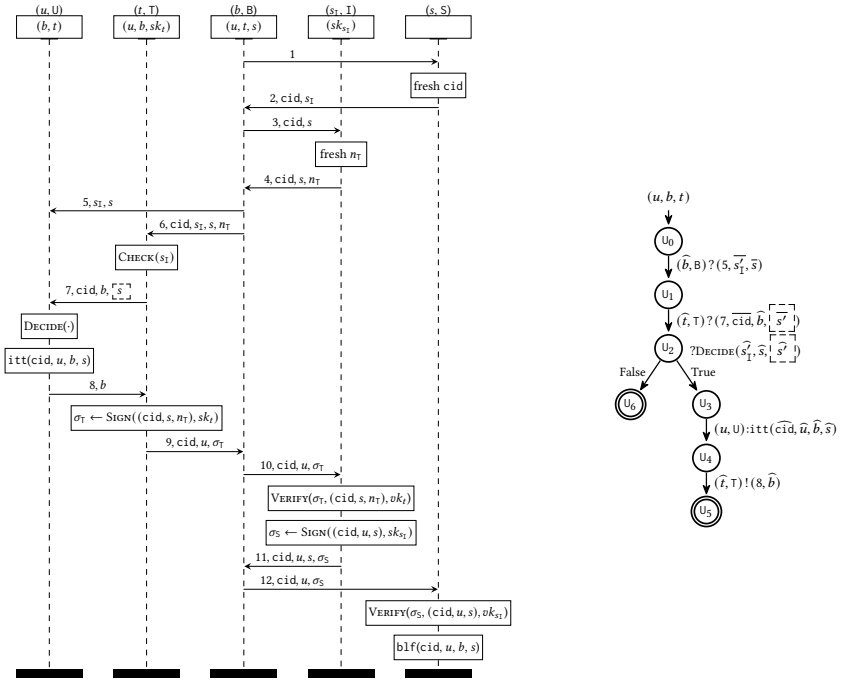


Figure 1: Web Authentication-based eID WebAuthn: MSC and user-role state machine

induced by the rule

$$\frac{l \,-[\, a \,]\!\rightarrow r \in_E \text{ginsts}(R) \qquad \text{lfacts}(l) \subseteq^{\#} S \qquad \text{pfacts}(l) \subseteq S}{S \xrightarrow{\text{set}(a)}_R \left(\left(S \setminus^{\#} \text{lfacts}(l)\right) \cup^{\#} r\right)}$$

where the functor $\text{set}(\cdot)$ maps multisets to sets. This rule describes when a state S can change to a state S'. The first condition $l \,-[\, a \,]\!\rightarrow r \in_E \text{ginsts}(R)$ identifies a rule that shall be applied. Its premises l need to be in the state S, where linear facts need to be contained as a multiset, while persistent facts only need to be contained as a set. Linear premise facts are removed from the state and the conclusions r are added to the state, so that S' must be $\left(\left(S \setminus^{\#} \text{lfacts}(l)\right) \cup^{\#} r\right)$.

The state machine in Fig. 1 has multiple transitions, but usually one rule includes all received messages as premises and sent messages as conclusions. The user's behaviour can in fact be modelled with a single rule $l \,-[\, a \,]\!\rightarrow r$:

$$l = \{\mathsf{U}_0(x_u, x_b, x_t), \mathtt{Msg}(x_b, x_u, (5, c_1, x_i, x_{s_1})), \mathtt{Msg}(x_t, x_u, (7, c_2, x_b, x_{s_2}))\}$$
$$a = \{\textsc{Decide}(x_u, x_{s_1}, x_{s_2}, x_i, c_1, c_2), \mathtt{itt}(c_2, x_u, x_b, x_{s_1})\}$$
$$r = \{\mathtt{Msg}(x_u, x_t, (8, c_2, x_b))\}$$

Intuitively, the rule models the reception of Messages 5 and 7 in Fig. 1. The actions denote both the decision event and the intent event. The only conclusion is Message 8. Since the user agent will not do anything else, we may omit its last state. We will model different user mistakes by restricting when $\textsc{Decide}(\cdot)$ is allowed to return $\mathtt{True}$ (e.g. that x_i is the only identity provider). Note that we use variables in the rules, e.g. $\mathsf{U}_0(x_u, x_b, x_t)$ rather than names as in $\mathsf{U}_0(u, b, t)$. This makes the rule applicable to any agent, not just the agent named u.

The server in Fig. 1 generates a fresh nonce to prevent adversaries replaying the identity provider's signature. Freshness is modelled with a rule $-[\,]\!\rightarrow \mathsf{Fr}(n)$ that has no premise and $\mathsf{Fr} \in \Sigma_{Fact}$ is linear. Executions are restricted such that n only occurs once in a rule application. Schmidt et al. [22] explain the details.

We get a transition system with the empty multiset $\emptyset^{\#}$ as its initial state. A set of rules essentially defines a protocol and its adversarial environment. An *execution* e of a protocol is an alternating sequence of states and sets of actions $S_0, a_1, S_1, \ldots S_{n-1}, a_n, S_n$ such that $S_0 = \emptyset^{\#}$, and for every $i \in \{1, \ldots, n\}$ we have that $S_{i-1} \xrightarrow{a_i}_P S_i$. The *trace* of the execution is $\text{trace}(e) = a_1 \ldots a_n$, the sequence of actions. We are particularly interested in the events in traces. In our setting, the actions of every transition will have at most one event.

2.1.3 Adversarial capabilities. The adversary's basic capabilities are also modelled by rules and special facts. The facts $\mathsf{K}(x)$, $\mathsf{Out}(x)$ and $\mathsf{In}(x)$ express that the adversary, respectively, *knows*, *learns* (gets an output) and *sends* (creates an input) x. The corresponding rules are $\mathsf{Out}(x) \,-[\,]\!\rightarrow \mathsf{K}(x)$ and $\mathsf{K}(x) \,-[\,\mathsf{K}(x)\,]\!\rightarrow \mathsf{In}(x)$. The adversary knows any public value (rule $-[\,]\!\rightarrow \mathsf{K}(x{:}pub)$), can make up fresh names (rule $\mathsf{Fr}(x{:}fr) \,-[\,]\!\rightarrow \mathsf{Fr}(x{:}fr)$) and can construct new messages (rules $\mathsf{K}(x_1), \ldots, \mathsf{K}(x_n) \,-[\,]\!\rightarrow \mathsf{K}(f(x_1, \ldots, x_n))$ for any $f \in \Sigma_{Op}$ of arity n).

These basic capabilities are then augmented with protocol-specific capabilities. In our setting, we have rules that create agent names with given roles and specific initial knowledge and allow the adversary to corrupt some of these agents. Further rules allow the

adversary to see messages sent to corrupted agents and send messages on their behalf (using, respectively, Out and In). We also make sure that the adversary knows all verification keys.

2.2 eID Protocols

In an eID protocol, a *user* contacts a *server* (relying party) through a *browser tab*. The user signals their *intent* to log into a particular server via a given browser tab, while the server signals its *belief* about which user logged in via which browser tab. We use these signals to define security notions for eID protocol.

Section 2.1 explained how MSCs are translated into Tamarin rules. We now give further details on how we model eID protocols in Tamarin. Again, we use the eID protocol from Fig. 1 as an example. First of all, we point out that the MSC actually describes two variants of the protocol, WebAuthn that excludes the dashed boxes, and WebAuthn$^+$ that includes the dashed boxes.

As a rule, we do not model why the user tries to log into a server, since this varies greatly. We do not want the user to tell the browser tab which server to talk to, since this makes phishing attacks harder to model. Instead, a browser tab initiates the login process by sending a message to the server. It is not obvious that a user always has a clear idea of what website they are logging into. We model this by having the user infer the server identity from some received message (which may or may not be consistent with other received messages).

The Decide$(\cdot)$ function is left unspecified in the MSC. It models things that the user should verify before proceeding with the protocol run. The possibility that the user could stop is not visible in the MSC, which in some sense shows a "typical honest" run of the protocol. It is, however, visible in the state machine for the user rule but the Decide$(\cdot)$ function is left opaque. In the Tamarin model, we use preconditions that prescribe how users would behave given parameters for Decide$(\cdot)$, e.g. if they also accept a server that does not match the one from before. This provides a flexible way to study many different user behaviours.

A user communicates with browser tabs and is able to distinguish different browser tabs from each other. A browser tab is tied to a user and communicates with servers. Servers send messages to the browser tab that are either shown to the user in a message that includes trustworthy information (the server URL displayed by browser chrome) and untrustworthy information (that supplied by the server, the actual web page), or direct the browser tab to contact other servers. The browser tab forwards any input from the user to the server. We allow browser tabs to be individually corrupted, modelling the fact that users' browser tabs may be distributed over several distinct computers, only some of which are compromised.

Ideally, we would define common browser tab rules shared by all protocols. However, it is challenging to provide rules that are sufficiently general to express all the required behaviour, but also sufficiently limited for Tamarin to be able to work with them. For instance, WebAuthn uses a FIDO2-like token connected directly to the user's browser tabs, which is very different behaviour from ordinary browser functionality. Therefore, we model browser tabs as protocol-specific rules, but take care that adversarial behaviour is not unduly constrained.

We model the user's token as a distinct agent. A user's token will only communicate with browser tabs that communicate with the token's user.

In general, we distinguish between a role and an agent to play a role in a protocol: for instance, while U represents the user role, the identifier for an agent might be u. In general, there can be multiple agents who play the same role in the same protocol execution but their identifiers allow us to distinguish them.

Some of our protocols have other roles, which may either model infrastructure (common to all users, servers and executions) or per-user devices (tokens, apps, etc.). We do not restrict how these additional roles communicate. However, since we intend to model the real world, our infrastructure will only communicate with servers, browser tabs and some user devices. Servers will only communicate with browser tabs and infrastructure, and never directly with user devices. User devices will only communicate with the user or the user's browser tabs. Only browser tabs and user devices will communicate directly with the user.

We use secure channels to model communication between agents. Communication between electronic devices are typically through proper secure channels such as TLS, whose security properties match our modelling. Communication between the user and their devices happen through physical channels (such as a monitor and a pair of eyes), which we also model as secure channels.

2.3 Security

2.3.1 Rationale. As discussed in the introduction, we follow the eIDAS regulation [9] in defining security. The overall goal is to give the server confidence that it is actually talking to the claimed user. In other words, whenever the server finishes an execution of the protocol, we will need to map this one to an execution of the user role whose intent was to log in to this particular server.

The conversation that happens *after* the identity claim has been substantiated by the eID protocol is usually the actual object of interest, and the identification is an ancilliary (but necessary) process. Of course, the user and the server are not having a direct conversation, but instead the browser tab acts as an intermediary. It is then crucial for the effective security of an eID protocol that the server and user agree on which browser tab is acting as an intermediary.

Intuitively, if there is an event $\texttt{itt}(u, b, s)$, the user u intended to have a conversation with the server s via the browser tab b. Analogously, the belief event $\texttt{blf}(u, b, s)$ indicates that the server s believes it is having a conversation with the user u via the browser tab b.

Browser tab. We model the server as interacting with a browser tab on a computer. Our model decrees that all browser tabs are independent, and that no browser tab will run an eID protocol more than once. In the real world, a server is talking to a web browser running on a computer identified by an IP address that may be shared by many computers. We assume that the server and the web browser running on the computer communicate using TLS, but TLS as used by web browsers is usually unable to distinguish different browser tabs talking to the same server. In fact, it would probably be dangerous to assume that browser tabs on different computers sharing an IP address could be distinguished. However,

because TLS provides integrity and confidentiality, a server may embed secrets in the HTML code and redirects sent to a browser tab, and these secrets may later be used to distinguish one browser tab from another. Note that the server must be extremely careful in how these secrets are sent and stored within the browser. If these secrets are stored as cookies, they may leak from one browser tab to another, and one browser tab may inadvertently change the state of a different browser tab. Likewise, our model says that every browser tab can only run an eID protocol once. Again, this is not actually true, and if secrets are not carefully managed, this can reveal things about previous runs of the protocol, breaking security. *Developer mistakes as described in this paragraph cannot be modelled in our framework.*

Corruption. The adversary may corrupt agents. For the protocols we study in a Dolev-Yao model, adaptive corruption is not so interesting and significantly complicates definitions and analysis. Therefore, we use so-called *static corruption*, where the adversary has to decide at the start if an agent is going to be honest or corrupt. In our model, we explicitly model the creation of agents, and we fix on static corruption by requiring the adversary to either corrupt the agent or make it honest before the agent does anything. Hence, at the beginning of every execution of a protocol, we will assume a set of honest agents $\mathfrak{I}_{\mathcal{H}}$.

Note the adversary takes control of a corrupted agent, which is different from reveal queries (such as found in key exchange) that leak key material and other secrets. Once an agent is corrupted in our model, it is not constrained to follow the role script and cannot emit events that contribute to security claims.

We have chosen not to allow corrupt users. We can design protocols where corrupt users give the adversary additional powers because they have access to shared secrets (e.g. a secure token contains a shared secret key which is used for authentication). This is not the case for our protocols. So far, we have also disallowed infrastructure corruption, but we refer to a discussion in Section 5.1.

The various tokens are also interesting from a corruption perspective, as resistance to corruption is the whole point of such tokens. Tokens might be connected to a computer via a USB cable or a Bluetooth wireless connection, and in this case it is argued that the software on the token is sufficiently simple that we can rule out adversarial compromise. Sometimes the token is not connected to anything at all, and in such cases corruption is unreasonable. A slightly different case concerns tokens that are owned by corrupted users (which we disallow). Such tokens would have to withstand much stronger attacks. This would be a particular concern if the token contains a system secret, and not just per-user secrets. But even if the token only contains per-user secrets, it is possible to design eID protocols that break when tokens are corrupted, but not when tokens are honest. However, we expect that for most real-world protocols, disallowing token compromise is a reasonable assumption.

We assume that the identity provider is honest. With the secure channel assumption, we hence need not consider how to protect (hashing, salting, etc.) passwords stored at the infrastructure provider.

2.3.2 Definitions. A standard way to phrase authentication properties [17] is via variants of *agreement*. This strategy also applies to the security of eID protocols. In order to emphasise this, we begin by defining a version of injective agreement in our model that allows for a sufficiently flexible specification of exactly what is agreed upon. We also need the events to appear in the correct order.

Definition 1. *Let $w = w_1 \dots w_n$ be a trace of an eID protocol execution. We say that w satisfies* injective ordered agreement *with respect to a relation R on events with labels* $\mathtt{ev}$ *and* $\mathtt{ev}'$*, iff for the two index sets $I = \{k \mid 1 \le k \le n \wedge w_k = \mathtt{ev}(x_1, \dots, x_j)\}$, and $I' = \{i \mid 1 \le i \le n \wedge w_i = \mathtt{ev}'(y_1, \dots, y_t)\}$, there exists an injective mapping $f : I \to I'$ such that if $f(k) = i$, then $i < k$ and $w_k \, R \, w_i$.*

We define security notions for eID protocols by defining relations on events, i.e. by instantiating R from the above definition in different ways. The strongest security notion (referred to as "security") says that the server belief and the user intent should agree on server, user and the browser tab intermediary. If the user communicates with a corrupt browser tab, this cannot be guaranteed, so we allow the two browser tabs to be different as long as *both* are corrupt.

The second, weaker security notion ("weak security") captures the case where the server and the user agree on the server and the user, but need not agree on the browser tab. This is weaker, because we now allow this to happen even if the user (or server) is communicating with an honest browser tab. This can happen when the user's actions verify the server identity, but are insufficiently tied to the browser tab.

The final security notion ("recent aliveness") captures the case where the server and the user only agree on the fact that the user tried to log in somewhere. In other words, the user must have been active in order for the login to happen. This happens when the user's actions do not verify the server identity.

Definition 2. *Define relations R_S, R_W and R_R by $\mathtt{blf}(u, b, s)$ R_S $\mathtt{itt}(u', b', s')$ iff $u = u'$, $s = s'$ and either $b = b'$ or $b, b' \notin \mathfrak{I}_{\mathcal{H}}$; $\mathtt{blf}(u, b, s)$ R_W $\mathtt{itt}(u', b', s')$ iff $u = u'$ and $s = s'$; and $\mathtt{blf}(u, b, s)$ R_R $\mathtt{itt}(u', b', s')$ iff $u = u'$. Let $w = w_1 \dots w_n$ be a trace of an eID protocol. If w satisfies injective ordered agreement with respect to R_S (resp. R_W, R_R), we say that w satisfies* security *(resp.* weak security, recent aliveness*).*

We have found it useful to consider the security of protocols in a setting where the user only communicates with honest browser tabs. While this is in general not a defensible setting, it may sometimes be justifiable, and we have found that in this rather narrow context, there is an aspect that the above notions miss. The idea is related to the idea that usually, the object of interest is the conversation that follows electronic identification, with the identification being ancillary. In that case, when (weak) security fails, it matters hugely who is in control of the subsequent conversation; we introduce a final notion ("honesty-preservation") to capture this setting.

Definition 3. *Let $w = w_1 \dots w_n$ be a trace of an eID protocol. We say that w satisfies* honesty-preservation *if for any event $w_i = \mathtt{blf}(u, b, s)$, we have $b \in \mathfrak{I}_{\mathcal{H}}$.*

2.3.3 Technical extension. The standard way to express injectiveness in Tamarin is as agreement over some fresh value. We therefore define a technical extension of injective agreement considering variants of the relations we used to define security. The idea is for the

server role to generate a fresh nonce and require that the user and the server also agree on this nonce (in addition to satisfying the relation). Since fresh nonces are unique, this immediately implies injectivity. We then only need to prove the existence of preceding matching intent events, which is much easier to express in Tamarin.

For technical reasons, we need two more events similar to `blf` and `itt`, except they include the nonce `cid`. The events will have the labels $\texttt{blf}_0$ and $\texttt{itt}_0$. The intent event $\texttt{itt}_0$ follow immediately after the event `itt`, while the belief event $\texttt{blf}_0$ must immediately precede the event `blf`. This ensures that if `blf` happens, then $\texttt{blf}_0$ has happened, and if $\texttt{itt}_0$ happens, then `itt` has also happened.

Definition 4. *Consider an eID protocol where the server role generates a fresh nonce* `cid`. *Let $w = w_1 \dots w_n$ be a trace of of this eID protocol. We say that w satisfies* unique ordered agreement *with respect to R on events with labels* `ev` *and* `ev'`, *iff for the two sets $I_0 = \{(k, \texttt{cid}_k) \mid 1 \le k \le n \wedge w_k = \texttt{ev}_0(\texttt{cid}_k, x_1, \dots, x_j)\}$, and $I_0' = \{(i, \texttt{cid}_i) \mid 1 \le i \le n \wedge w_i = \texttt{ev}_0'(\texttt{cid}_i, y_1, \dots, y_t)\}$ there exists an injective mapping $f_0 \colon I_0 \to I_0'$ such that if $f_0(k, \texttt{cid}_k) = (i, \texttt{cid}_i)$, then $\texttt{cid}_k = \texttt{cid}_i$, $i < k$ and if $w_k = \texttt{ev}_0(\texttt{cid}_k, x_1, \dots, x_j)$ and $w_i = \texttt{ev}_0'(\texttt{cid}_i, y_1, \dots, y_t)$ then $\texttt{ev}(x_1, \dots, x_j)\ R\ \texttt{ev}'(y_1, \dots, y_t)$.*

Remark 1. Note that the protocol contains events that are slightly different from the events R relates, in the sense that the former mention the nonce, while the latter do not. This is intentional. A relation R' on the events used in the protocol is defined implicitly, and the definition amounts to agreement relative to this relation. The fresh nonce then ensures injectivity.

Expressing unique ordered agreement for our relations is straightforward in Tamarin. When Tamarin proves that our relation holds, it also follows that injective agreement holds for the same relation.

Theorem 1. *Consider an eID protocol where the server role has a fresh nonce* `cid` *and contains the events* $\texttt{blf}_0$ *and* $\texttt{itt}_0$ *as discussed above. If the protocol satisfies unique ordered agreement with respect to a relation R, then it satisfies injective ordered agreement with respect to the same relation.*

Proof. We consider a trace that satisfies unique ordered agreement, and let $f_0 \colon I_0 \to I_0'$ be some injective mapping satisfying the requirements. We describe how to construct the injective mapping $f \colon I \to I$.

Let $k \in I$. By construction of the protocol, there exists a corresponding $(k_0, \texttt{cid}_{k_0})$ in I_0 belonging to the same server instance. By unique ordered agreement, there exists a $(i_0, \texttt{cid}_{i_0})$ in I_0' such that $f_0(k_0, \texttt{cid}_{k_0}) = (i_0, \texttt{cid}_{i_0})$ and $i_0 < k_0$. Again, by construction of the protocol, there exists a corresponding $i \in I'$ belonging to the same user instance. We then define $f(k) = i$.

The function f is now defined for all $k \in I$, and the function satisfies the requirements of injective ordered agreement. □

We note that no real-world protocol will display a full nonce to the user, and no real user will ever take such a nonce into account. The user knowing `cid` is in this sense "synthetic", not real. This means that the analyst needs to be extremely careful when modelling eID protocols, so that the synthetic nonce `cid` is not used to achieve security in a way that does not model reality.

In order not to inadvertently achieve unrealistic security properties, we include the `cid` in only one message sent to the user in each protocol, and ensure the user never forwards the `cid` to any agents. However, in order to avoid artificial attacks such as the adversary generating a fresh `cid` via a compromised agent (thereby breaking unique ordered agreement since the user and server have distinct `cid` values, but not injective ordered agreement), we need to be careful when choosing which agent shares the `cid` with the user. Since we take authentication tokens to be trusted (non-corruptable) agents, they make for a natural choice to include the `cid` in their first message to the user, when these tokens are available.

This explains that unique ordered agreement is harder to achieve than injective ordered agreement: It is easy to build models susceptible to artefact attacks that do not model reality. For instance, one can model protocols that allow an adversary can change the nonce `cid` between the user and the server in a trivial way, which breaks unique ordered agreement, but not injective ordered agreement.

3 Protocol Specifications and Analysis

In this section, we provide details on the protocol specifications we have analysed using our framework. A summary of the results can be found in Table 1. The Tamarin models are available in a public repository[6].

3.1 Password Only

In the password-only protocol Password (see Fig. 2), the server redirects the browser tab to the identity provider, which asks the user for their password. The user verifies the identity provider's URL and enters the password. The identity provider checks the password, and then generates a signature on a message containing the server's challenge nonce, the user's identity and the server's identity. It sends the signature to the server by browser tab redirection.

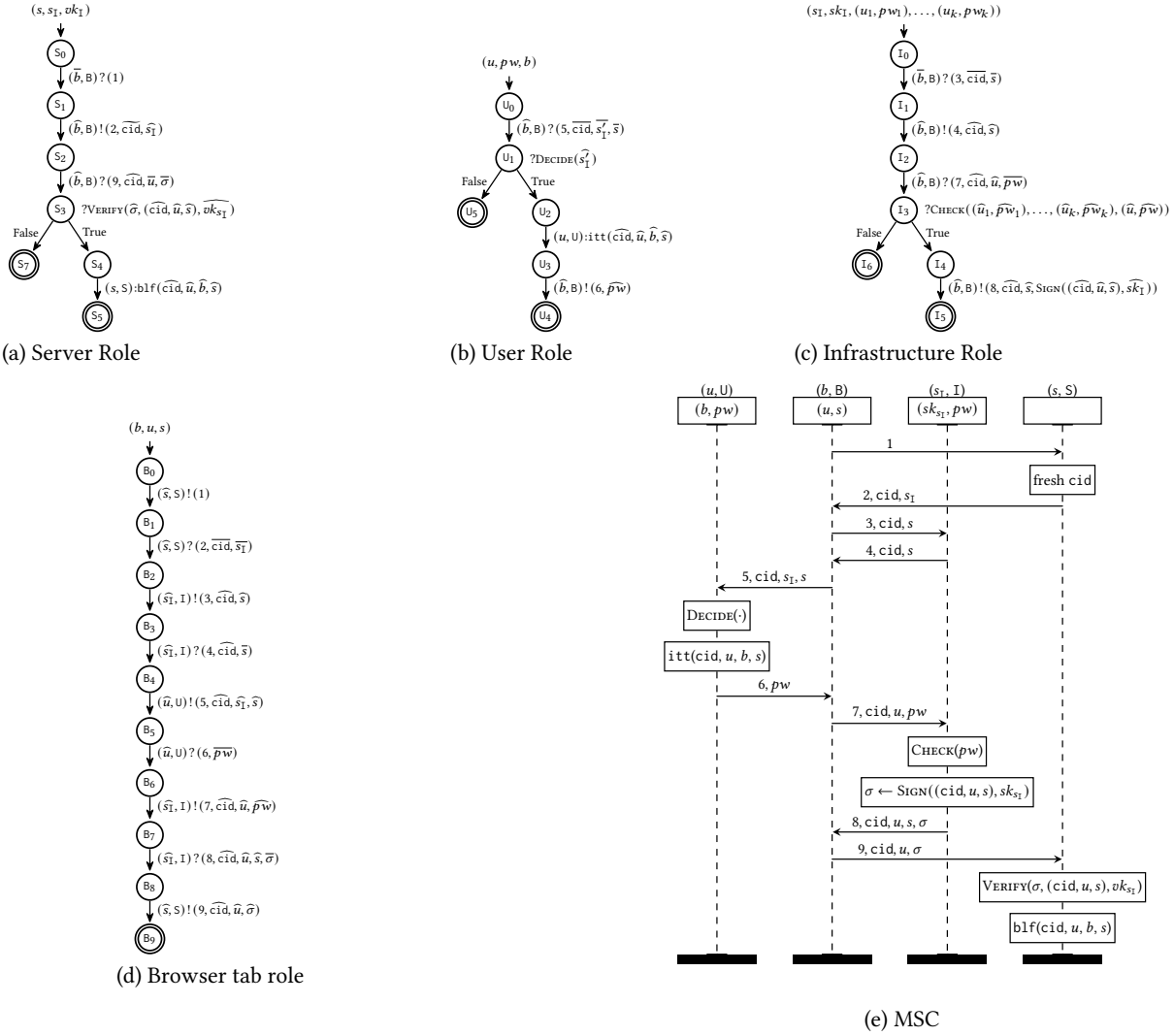
Analysis. If the user communicates with a corrupt browser tab, the adversary learns the password. If the user is imperfect, a phishing attack will relieve the user of their password. In either case, the protocol has no security.

If the user verifies the URL and never uses corrupt browser tabs, the password stays secret and the protocol achieves security. Users that are very careful to update their computers and not install malware, use password management software and are not otherwise targeted by sophisticated adversaries, may approximate this setting.

3.2 App Only

The app (running on some personal device, usually a phone) communicates directly with the identity provider. In the most basic variants, the user simply accepts the login by pressing a button. In the app only protocol App, the server redirects the browser tab to the identity provider, which asks the user to approve the login. The user verifies the identity provider's URL (treated as a global constant in the Tamarin code, instead of as part of the user's state, since we only model one identity provider). Simultaneously, the provider contacts the user's app. After the user approves, the identity provider generates a signature (as in Password), and sends it to the server by redirecting the browser tab. The MSC can be found in Fig. 3.

[6] https://github.com/festutz/eID-protocol-models



(a) Server Role

(b) User Role

(c) Infrastructure Role

(d) Browser tab role

(e) MSC

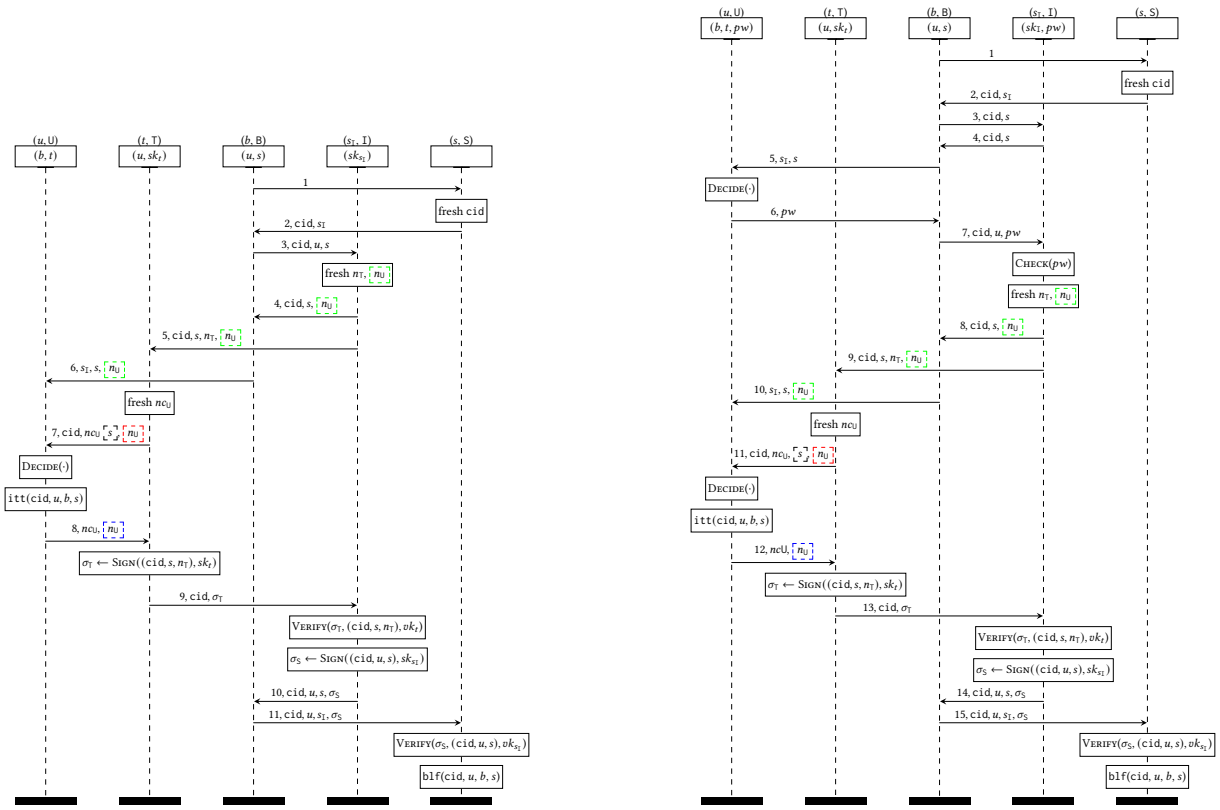
Figure 2: State machines and MSC for a password-based eID protocol, Password.

A real-world app would refuse parallel sessions. Our model allows this, which requires some care when modelling to avoid spurious attacks. One problem is that the approval in the app is independent of the login process. One option (App Cmp) is that the identity provider generates a nonce (usually a short number) and shows it to the user via both the browser tab and the app, who is asked to verify that the nonces match (or, better, requires the user to identify the correct nonce among several displayed ones). Another option (App Write) is that the nonce is shown in the browser tab only and the user enters it into the app.

A second problem is that the approval in the app does not show any information about the server identity. In the alternative protocols (marked with $^+$), the identity provider sends the server identity to the app, which displays it to the user. The user verifies the server identity displayed by the app.

Analysis. The basic app only protocol App achieves recent aliveness, since the identity provider ensures that it interacts with the app for each login. However, since the app does not include any information, the adversary can start a malicious login process and trick the user into approving it. If the app displays the server name and a careful user verifies the name (perfect user), the protocol achieves weak security (the adversary can still trick the user into approving a malicious login process instead of their own honest login process). Hence, these techniques can successfully tie the app to the browser tab, *provided that the user behaves perfectly*.

For the protocol App Cmp$^+$, we did a more fine-grained study of the possible user mistakes. There are three things the user should verify, giving us eight different possibilities for combinations of mistakes. As we see from the diagram in Table 1, verifying the server identity alone is sufficient for weak security. We believe that this knowledge could be used to design better user interfaces.



(a) App

(b) Two-factor

Figure 3: MSC for eID protocols using an app only (App), and an app in conjunction with a password (Two-factor)

3.3 Twofactor – Password + App

One can also require a password and the use of an app. In our version, the identity provider asks for the password before communicating with the token. As for the app only protocol, we get 6 variants of this protocol. The MSC can be found in Fig. 3.

Analysis. We see that the effect of the password is fairly minimal. Since the password leaks if the user communicates with a corrupt browser tab, or through a phishing attack if the user is not sufficiently careful, it is not surprising that the variants of Two-factor are not more secure than App.

However, it is slightly surprising that when the user never communicates with corrupt browser tabs and is otherwise perfect, the protocol seems to have *less* security than the password-only protocol Password. The reason is that the user's intent is only signaled after the user decides to approve in the app. In other words, the user may change their mind about logging in, even after entering the password. This allows an attack where the user initiates two login attempts in parallel, but confuses which browser tab the approval is made for. For Two-factor$^+$, the two login attempts must be made to the same server, making the protocol honesty-preserving, better than the corresponding app-only protocols. While these attacks are real, we should not overestimate their importance.

We emphasise that in a setting where the user never communicates with corrupt browser tabs and is otherwise perfect, the protocols with passwords are actually better than the app-only protocols. The reason is that we model the user as making the login decision just before accepting in the app, and not before entering the password. The consequence is that the user may log into one browser tab, but believe they did it in another browser tab. There are realistic settings where this difference will show itself, which is why we emphasise these partial results. We also refer to Section 5.1 for a discussion of relevant future work.

3.4 Web Authentication – Token only.

WebAuthn, the Web Authentication protocol we consider, uses a FIDO2-like token to identify the user to the identity provider, who then vouches for the user to the server. The MSC can be found in Fig. 1 (p. 4). A FIDO2-like token is a small device, hardened against physical and logical attacks, that locally connects to the user's computer. There are other options, including using hardened computer subsystems.

Like the app case, our model allows the token to run multiple sessions in parallel, which is not necessarily realistic. We assume that the user interface makes it clear which browser tab is talking to the token at any one point. The simplest tokens do not have a

display, but more advanced tokens could display information about the server identity, giving us the WebAuthn$^+$ protocol.

Analysis. The situation for WebAuthn and WebAuthn$^+$ is simple. Both protocols achieve a minimum of recent aliveness, and security otherwise: WebAuthn is vulnerable to corrupt browser tabs, while WebAuthn$^+$ is only vulnerable to corrupt browser tabs in conjunction with careless users.

4 Related Work

As mentioned, authentication protocols employed in practice have typically been analysed heuristically, with a focus on identifying and analyzing best-known attacks. Worse, models often differ in subtle but important ways between authors, making direct comparisons of protocols difficult.

Many authentication protocols *have* been formally analysed though, e.g. the handshake protocol of TLS 1.3 [7]. Barbosa et al. [1] formalise and give security proofs for FIDO2-related protocols in the cryptographic model, not a Dolev-Yao model. Relative to eID protocols, their security notions are limited and many attacks and settings cannot be expressed in their model.

One effort to evaluate different login schemes, including federated login systems, is due to Bonneau et al. [3]. Their framework was not intended for a formal security analysis as such; security is just one component in their approach and is quite ad-hoc, classifying systems only with respect to various standard attack vectors, such as shoulder-surfing, phishing attacks, theft-of-token and more.

Ellison's ceremonies [11] have received due attention in the study of human participation in security protocols. While intuitive, formalising the ideas has proven challenging. Basin et al. [2] gave a simple model of human behaviour, and analysed a selection of human-to-server authentication protocols via Tamarin (but not federated login systems). Their security notion is close to "recent aliveness" from Cremers and Mauw [5] and Definition 2. Fila and Radomirović [13] gave a more complex model of human interaction; while their model can be expressed in Tamarin, the authors doubt Tamarin will be helpful to their context.

Instead of modelling the user in all their human complexity, one may consider the class of all errors the user could possibly make during the execution of the protocol. From this, we essentially get a collection of protocols that differ only in the user's behaviour. This is the direction that was taken by Gjøsteen [14], who applied the model to federated login systems in Norway. A more sophisticated example of the same approach is due to Jacomme and Kremer [16]. They study user mistakes, as well as different kinds of malware compromise, but do not consider users with some honest and some corrupted computers. Even though they study eID protocols, they only study them in a very simplified setting where the identity provider and its interactions are elided. Paier et al. [20] extend the model to analyse a concrete Italian eIDAS-compliant eID protocol.

Mainka et al. [18] survey previous attacks on single sign-on systems and study how they apply to the then-current OpenID standard and its implementations. The evaluation of existing implementations for security against a collection of known attacks is automated, but this is not an attempt to prove the protocol secure. They do consider multiple identity providers, which we do not.

Engelbertz et al. [12] use the same methodology to evaluate eIDAS federated login systems, including the complexity of cross-border use.

5 Conclusion and Future Work

We have presented the first framework for eID protocols that combines automated analysis, fine-grained control over corruption of agents, and the ability to model user mistakes. Our notion of eID protocol captures a wide array of different protocols deployed in practice. We considered 15 variations of 4 such protocols, i.e. password only, app only, multi-factor, and the FIDO2-like WebAuthn protocol. Using Tamarin, our framework was able to automatically analyse different scenarios of user mistakes and corruption for these models. It also allows us to give flexible definitions of security that are consistent with the eIDAS regulation, capturing the wide spectrum of security guarantees provided by today's deployed protocols. As expected, we found both variations of WebAuthn to provide the strongest security guarantees among all protocols considered, and recommend continued adoption. Our framework also gives means for fine-grained studies of user mistakes. Such an analysis does not only give a lattice of security guarantees but also insights on which user checks are most important. We believe this could be beneficial for the design of better user interfaces. For detailed results of both types of study, we refer to Table 1.

5.1 Future Work

We leave a number of open problems unresolved. From the list below, some items are not conceptually difficult, but will still require some effort to carry out.

User mistakes. We have chosen not to model every reasonable type of user mistake. One interesting case is where the adversary gets access to the user's app or token (storing a token in an unlocked drawer in an office; registering a partner's biometrics on your phone; etc.). Alternatively, the user may interact with the app or token outside of login attempts (such as accepting an app verification request just to make the notification go away). Yet another "mistake" is allowing someone to look over your shoulder as you type your password (shoulder surfing). Finally, users sometimes simply give away their password (password sharing is fairly common among couples, for example). Some protocols are more vulnerable to these mistakes than others. The effects are obvious and deleterious. These mistakes can be modelled, and Tamarin shall be able to find the relevant attacks (at some extra computational cost). We expect that the most interesting result would be to better separate the app-based and two-factor protocols in settings where the adversary is otherwise restricted. Such settings are realistic, and our present results should not be taken as evidence that passwords may just as well be removed from authentication protocols involving apps or tokens.

Dishonest users and infrastructures. Modelling multiple infrastructures sharing a single protocol is interesting, as it introduces some interesting new attacks. It is also interesting to study what happens when the infrastructure is corrupted. Obviously, in such cases, we cannot achieve perfect security, but for some protocols we can distinguish logins made by a corrupt infrastructure from

logins made by honest users, and this is interesting to capture in a model. We would do this by having the server claim a piece of *evidence* as part of its belief event, and then use this evidence to distinguish an honest user from a corrupt infrastructure.

Electronic signatures and European Digital Identity Wallets. We have studied electronic identification, but the eIDAS regulation also includes electronic signatures, and more recently a wallet that can be used to selectively disclose attributes. Quite a lot of work has been done on the cryptographic underpinnings of wallets (including linkability and quantum-safe instantiations), but studying the protocols for making electronic signatures and disclosing attributes remains interesting.

References

[1] Barbosa, M., Boldyreva, A., Chen, S., Warinschi, B.: Provable security analysis of FIDO2. In: CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 125–156. Springer (2021)

[2] Basin, D.A., Radomirovic, S., Schmid, L.: Modeling human errors in security protocols. In: CSF 2016 Computer Security Foundations Symposium. pp. 325–340. IEEE (2016)

[3] Bonneau, J., Herley, C., van Oorschot, P.C., Stajano, F.: The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In: 2012 IEEE Symposium on Security and Privacy. pp. 553–567. IEEE (2012)

[4] Chadwick, D.W.: Federated Identity Management, p. 96–120. Springer-Verlag, Berlin, Heidelberg (2009), https://doi.org/10.1007/978-3-642-03829-7_3

[5] Cremers, C., Mauw, S.: Operational Semantics and Verification of Security Protocols. Information Security and Cryptography, Springer Berlin, Heidelberg (2012)

[6] Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–208 (1983). doi:10.1109/TIT.1983.1056650

[7] Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol. Journal of Cryptology **34**(4), 37 (2021)

[8] Dreier, J., Hirschi, L., Radomirovic, S., Sasse, R.: Automated unbounded verification of stateful cryptographic protocols with exclusive OR. In: CSF 2018 Computer Security Foundations Symposium. pp. 359–373. IEEE (2018)

[9] Regulation (EU) no 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC (2882014), https://eur-lex.europa.eu/eli/reg/2014/910/oj/eng

[10] Regulation (EU) 2024/1183 of the European Parliament and of the Council of 11 April 2024 amending regulation (EU) no 910/2014 as regards establishing the European Digital Identity Framework (3042024), https://eur-lex.europa.eu/eli/reg/2024/1183/oj

[11] Ellison, C.: Ceremony design and analysis. Cryptology ePrint Archive, Report 2007/399 (2007)

[12] Engelbertz, N., Erinola, N., Herring, D., Somorovsky, J., Mladenov, V., Schwenk, J.: Security analysis of eIDAS – the cross-country authentication scheme in Europe. In: 12th USENIX Workshop on Offensive Technologies (WOOT 18). USENIX Association (2018)

[13] Fila, B., Radomirovic, S.: Nothing is out-of-band: Formal modeling of ceremonies. In: CSF 2024 Computer Security Foundations Symposium. pp. 464–478. IEEE (2024)

[14] Gjøsteen, K.: Protocol variants and electronic identification. Cryptology ePrint Archive, Report 2013/329 (2013)

[15] Hodges, J., Jones, J., Kumar, A., Jones, M., Lundberg, E.: Web authentication: An API for accessing public key credentials - level 2. W3C recommendation, W3C (Apr 2021), https://www.w3.org/TR/2021/REC-webauthn-2-20210408/

[16] Jacomme, C., Kremer, S.: An extensive formal analysis of multi-factor authentication protocols. In: CSF 2018 Computer Security Foundations Symposium. pp. 1–15. IEEE (2018)

[17] Lowe, G.: A hierarchy of authentication specification. In: 10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA. pp. 31–44. IEEE (1997)

[18] Mainka, C., Mladenov, V., Schwenk, J., Wich, T.: SoK: Single sign-on security - an evaluation of OpenID connect. In: 2017 IEEE European Symposium on Security and Privacy. pp. 251–266. IEEE (2017)

[19] Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 696–701. Springer (2013)

[20] Paier, M., Van Eeden, R., Miculan, M.: Formal analysis of multi-factor authentication schemes in digital identity cards. In: Software Engineering and Formal Methods: 22nd International Conference, SEFM 2024. p. 423–440. Springer-Verlag (2024)

[21] Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Aug 2018). doi:10.17487/RFC8446, https://www.rfc-editor.org/info/rfc8446

[22] Schmidt, B., Meier, S., Cremers, C.J.F., Basin, D.A.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: CSF 2012 Computer Security Foundations Symposium. pp. 78–94. IEEE (2012)