Computers & Security 140 (2024) 103771

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Enforcing security policies on interacting authentication systems

Francesco Buccafurri [a,*], Vincenzo De Angelis [b], Sara Lazzaro [a], Andrea Pugliese [b]

[a] *DIIES Dept., University Mediterranea of Reggio Calabria, Via dell'Universita 25, Reggio Calabria, 89124, Italy*
[b] *DIMES Dept., University of Calabria, Via Pietro Bucci, Rende, 87036, Italy*

ARTICLE INFO

Keywords:
Authentication
Security policies
Digital identity

ABSTRACT

Security policies of authentication systems are a crucial factor in mitigating the risk of impersonation, which is often the first stage of advanced persistent threats. Online authentication systems may often interact with each other, due to various mechanisms, such as account recovery or federated authentication. This leads to an implicit extension of the security policies of an authentication system with policies over which the system has no control. As a result, an authentication system that adopts very strong security policies can be unexpectedly weak. This paper deals with the above problem, which affects most real-world online authentication systems. The paper proposes a theoretical framework that formalizes authentication policies and interactions among authentication systems, together with a protocol that prevents, whenever an interaction is established or updated, the security issues described above. An SSI-based implementation of the proposed protocol is presented as well.

1. Introduction

Security policies of authentication systems play a crucial role in protection against attacks on impersonation (Bonneau et al., 2012; Campobasso and Allodi, 2020; Squicciarini et al., 2007). The latter is one of the most serious threats, as it can represent the basis of complex attack vectors, leading to advanced persistent threats (Alshamrani et al., 2019; Li et al., 2018; Oberle et al., 2016).

Nonetheless, in real-world online authentication systems, a serious problem exists that can invalidate the effectiveness of authentication-policy management, leading to paradoxical situations.

We illustrate this problem by considering a simple example. `Alice` registers with a powerful provider's cloud service called `inventedgiant.com`. `inventedgiant.com` is very attentive to security aspects, so it sets very strong security policies, such as multi-factor authentication, strong security rules for passwords, and throttling mechanisms. As usual, an account recovery mechanism is also provided, based on sending a reset link to an email address specified by the user. `Alice` uses the email address alice@inventedcolander.com. However, `inventedcolander.com` is somewhat distracted about security issues, so its security policies are quite weak. For example, users of its email service can set weak passwords like "Alice", "Bob", "Password", etc., and the system does not protest. `Alice` is not fully aware of security issues, and, on the other hand, does not give much importance to the email service by `inventedcolander.com`, which she uses just for (apparently) non-critical activities. Therefore, when, in 2022, she registered with the email service of `inventedcolander.com`, she set "Alice2022" as her password. Unfortunately `Alice` has an enemy, `Trudy`, who is very eager to steal all of `Alice`'s pictures stored at `inventedgiant.com`. It is a matter of fact that it is unfeasible for `Trudy` to break the authentication process of `inventedgiant.com` to impersonate `Alice`, but it is a piece of cake to violate the account alice@inventedcolander.com, which, fortunately for `Trudy`, `Alice` has published in her public social network profile (thus simplifying the task of `Trudy`). This will allow `Trudy` to achieve the aimed goal.

This intuitive example is a clear proof of a practical yet serious problem that affects existing online services. The problem arises from the possible interaction between different authentication systems. Federated authentication exposes this problem as well, regardless of the technology adopted to implement it (Sakimura et al., 2014; Wilson and Hingnikar, 2019). In general, authentication systems cannot be always viewed as closed systems, in which it suffices to enforce one's own security policies to ensure that the security level obtained is consistent with those policies. Indeed, the interaction with other authentication systems may result in a sort of propagation of security policies and, consequently, to unexpected possible security violations.

* Corresponding author.
E-mail address: bucca@unirc.it (F. Buccafurri).

https://doi.org/10.1016/j.cose.2024.103771
Received 23 October 2023; Accepted 12 February 2024
Available online 23 February 2024

Thus, a research question arises: *How can we ensure that security policies for authentication are enforced over interacting authentication systems?* In more detail, viewing interaction as a *dependency* relationship between different authentication systems, the question becomes: *How can we guarantee that such dependencies can be established and maintained without violating the coherence of the respective security policies?*

Our paper studies this problem by formalizing the interaction between authentication systems in the general case and by proposing a protocol ensuring security policy coherence. Specifically, our main contributions are the following.

1. We provide a formal characterization of authentication security policies.
2. We provide a formal characterization of the interactions among authentication systems. In the above example, `inventedgiant.com` plays the role of the "main" system whereas `inventedcolander.com` plays the role of the "secondary" system – this is formalized as a *dependency* of the main system on the secondary system.
3. On the basis of a theoretical framework built around the notion of dependency, we provide an algorithm to verify whether the dependency of a system S on another system S' can be established and kept over time. Intuitively, the verification is successful only if the security policies of S are not weakened by S' (i.e., an adversary cannot gain advantage by using S' to authenticate on S) – we provide a formal security analysis for this purpose.
4. We define a protocol to enforce the verification described above and we design a Self-Sovereign-Identity (SSI) based (Mühle et al., 2018; Preukschat and Reed, 2021) solution to implement this protocol. Our implementation considers a reference set of authentication attributes derived from the *OWASP authentication cheat sheet* (OWASP, 2024) and *NIST Digital Identity Guidelines: Authentication and Lifecycle Management* (NIST, 2017a).

To the best of our knowledge, no previous work deals with the problem considered in this paper, despite its practical relevance.

The remainder of the paper is organized as follows. In Section 2, we propose a model for authentication systems and their security policies. Section 3 discusses the registration of a user with a single authentication system. The interactions among different authentication systems are formalized in Section 4. Section 5 provides a formal security analysis of authentication in our proposed framework. An implementation of our proposal is described in Section 6. Finally, Section 7 discusses related work and Section 8 outlines our conclusions and identifies open challenges.

2. Modeling authentication systems

In this section, we provide our formal characterization of authentication systems. We assume the existence of a set $\mathcal{S}$ of systems and a set $\mathcal{U}$ of users.

2.1. Authentication attributes

In our framework, $\mathcal{A}_{auth} = \mathcal{A}_{usr} \cup \mathcal{A}_{sys}$ is the set of *authentication attributes*, where $\mathcal{A}_{usr}$ and $\mathcal{A}_{sys}$ are sets of *user* and *system* attributes, respectively. Each attribute $A \in \mathcal{A}_{auth}$ has an associated sequence

$$values(A) = \langle v_1, \ldots, v_{|values(A)|} \rangle$$

of possible values, equipped with a total order $\prec$ such that $v_i \prec v_{i+1}$ for all $i \in [1, |values(A)| - 1]$.

Example 1. *Consider a scenario where* $\mathcal{A}_{usr} = \{A_1, A_2\}$ *and* $\mathcal{A}_{sys} = \{A_3, A_4, A_5, A_6\}$ *with*

- A_1 =*"Password length", values*$(A_1) = \langle 4, \ldots, 256 \rangle$;
- A_2 =*"No dictionary words used for passwords", values*(A_2) = ⟨*false, true*⟩;
- A_3 =*"No password character limit", values*(A_3) = ⟨*false, true*⟩;
- A_4 =*"Method for password reset", values*(A_4) = ⟨*Security questions, URL token, PINs, Offline methods*⟩;
- A_5 =*"Token duration (hours) for password reset", values*(A_5) = ⟨*false, true*⟩;
- A_6 =*"PIN length for password reset", values*$(A_6) = \langle 4, \ldots, 12 \rangle$.

We also write $\textit{min-val}(A) = v_1$, $\textit{max-val}(A) = v_{|values(A)|}$, $\textit{succ-val}(A, v_i) = v_{i+1}$, and $\textit{prec-val}(A, v_i) = v_{i-1}$.

Given an attribute $A \in \mathcal{A}_{auth}$ and a value $v \in values(A)$,

$$\textit{voided-by}(A, v) \subset \mathcal{A}_{auth}$$

denotes the set of attributes that cannot be used when value v is used for attribute A. For instance, "Token duration (hours) for password reset" $\in$ *voided-by*("Method for password reset", PINs) and "PIN length for password reset" $\in$ *voided-by*("Method for password reset", URL token).

Finally, if $A \in \mathcal{A}_{sys}$, then

$$\textit{sys-val}(S, A) \in values(A)$$

denotes the value used by system S for attribute A.

2.2. System policies

Each system $S \in \mathcal{S}$ defines its *authentication policy* by specifying, for each attribute $A \in \mathcal{A}_{auth}$, a set

$$policy(S, A) = \{(m_1, C_1), \ldots, (m_n, C_n)\}$$

where, $\forall i \in [1, n]$:

- $m_i \in values(A)$;
- C_i is a set of pairs (A', v) with $A' \in \mathcal{A}_{auth}$, $A' \neq A$, $v \in values(A')$, and $v \neq \textit{min-val}(A')$.

Intuitively, the semantics of the policy is that $\forall i \in [1, n-1]$, if $\forall (A', v) \in C_i$ the value of attribute A' is at least v, then the minimum accepted value for attribute A is m_i.

Example 2 shows how our definition can express a policy which is recommended by NIST, along with a graphical representation.

Example 2. *NIST Special Publication 800-63B (NIST, 2017a) recommends the following:*

> *"Look-up secrets having at least 112 bits of entropy SHALL be hashed with an approved one-way function. Look-up secrets with fewer than 112 bits of entropy SHALL be salted and hashed using a suitable one-way key derivation function. The salt value SHALL be at least 32 in bits in length and arbitrarily chosen so as to minimize salt value collisions among stored hashes. Both the salt value and the resulting hash SHALL be stored for each look-up secret. For look-up secrets that have less than 64 bits of entropy, the verifier SHALL implement a rate limiting mechanism that effectively limits the number of failed authentication attempts that can be made on the subscriber's account."*

and

> *"Look-up secrets SHALL have at least 20 bits of entropy."*

We can express the above policy by making policy(S,"Bits of entropy for look-up secret") consist of the following pairs:

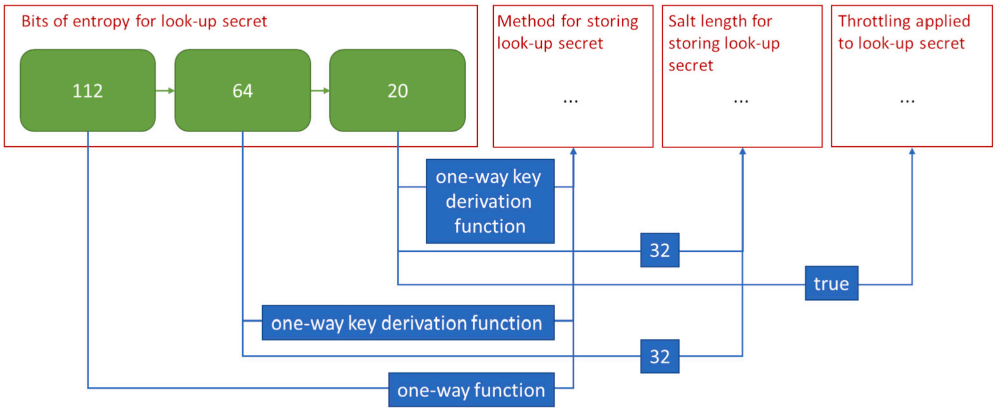
Bits of entropy for look-up secret
112
64
20
Method for storing look-up secret
...
Salt length for storing look-up secret
...
Throttling applied to look-up secret
...
one-way key derivation function
32
true
one-way key derivation function
32
one-way function

Fig. 1. Example system policy for attribute "Bits of entropy for look-up secret".

(112,	{("*Method for storing look-up secret*", *one-way function*)})
(64,	{("*Method for storing look-up secret*", *one-way key derivation function*), ("*Salt length for storing look-up secret*", 32)})
(20,	{("*Method for storing look-up secret*", *one-way key derivation function*), ("*Salt length for storing look-up secret*", 32), ("*Throttling applied to look-up secret*", *true*)})

Fig. 1 shows a graphical representation of the policy.

Given a policy $policy(S, A) = \{(m_1, C_1), \ldots, (m_n, C_n)\}$, we define its equivalent *expanded* version $exp\text{-}policy(S, A)$ as

$$\cup_{m^* \in [m_1, max\text{-}val(A)]} \{(m^*, C_1)\} \bigcup \cup_{i \in [2,n], m^* \in [m_i, prec\text{-}val(A, m_{i-1})]} \{(m^*, C_i)\}.$$

The equivalence will be clearer later on.[1]

Example 3. *In the case of Example 2, if we assume max-val("Bits of entropy for look-up secret") = 200, then the expanded version of policy(S, "Bits of entropy for look-up secret") consists of:*

- *a pair* $(m^*, \{(\text{"Method for storing look-up secret", one-way function})\})$ *for each* $m^* \in [112, 200]$;
- *a pair* $(m^*, \{(\text{"Method for storing look-up secret", one-way key derivation function}), (\text{"Salt length for storing look-up secret"}, 32)\})$ *for each* $m^* \in [64, 111]$;
- *a pair* $(m^*, \{(\text{"Method for storing look-up secret", one-way key derivation function}), (\text{"Salt length for storing look-up secret"}, 32), (\text{"Throttling applied to look-up secret"}, true)\})$ *for each* $m^* \in [20, 63]$.

2.3. Policy requirements

We now define a set of 6 requirements $policy(S, A)$ must satisfy.

R1. $\forall i \in [1, n-1]$, $m_i \succ m_{i+1}$.
R2. $\forall i \in [1, n]$, if $(A', v) \in C_i$, then $\nexists (A', \overline{v}) \in C_i$ with $v \neq \overline{v}$.
R3. If $A \in \mathcal{A}_{sys}$, then $sys\text{-}val(S, A) \geq m_n$.
R4. There do not exist a set $\{A_1, \ldots, A_k\}$ of attributes and a set $\{w_1, \ldots, w_k\}$ with $\forall j \in [1, k]$, $w_j \in values(A_j)$ such that:
- $\forall j \in [1, k-1]$, $(A_{j+1}, w_{j+1}) \in \overline{C}$ with $(w_j, \overline{C}) \in exp\text{-}policy(S, A_j)$;
- $(A_1, \overline{w}) \in \overline{C}$ with $(w_k, \overline{C}) \in policy(S, A_k)$ and $\overline{w} \succ w_1$.

R5. There do not exist a set $\{A_1, \ldots, A_k\}$ of attributes and a set $\{w_1, \ldots, w_k\}$ with $\forall j \in [1, k]$, $w_j \in values(A_j)$ such that:
- $\forall j \in [1, k-1]$, $(A_{j+1}, w_{j+1}) \in \overline{C}$ with $(w_j, \overline{C}) \in exp\text{-}policy(S, A_j)$;
- $(A_k, \overline{w}) \in \overline{C}$ with $(w_1, \overline{C}) \in policy(S, A_1)$ and $\overline{w} \prec w_k$.

R6. $\forall i \in [1, n]$,
- if $(A', v) \in C_i$, then $A' \notin V$;
- if $\exists\, p < i$ s.t. $(A', v') \in C_p$ and $A' \notin V$, then $(A', v) \in C_i$ with $v \geq v'$;

where $V = voided\text{-}by(A, m_i) \cup \bigcup_{(A'', v'') \in C_i} voided\text{-}by(A'', v'')$.

Intuitively, Requirements R1–R3 ensure that:

- The sequence of v_i values appears in decreasing order in $policy(S, A)$.
- An attribute cannot appear twice in the same C_i with different minimum values.
- If the policy is applied to a system attribute, then the lowest admitted value m_n is not higher than the value used by the system for that attribute.

Requirement R4 ensures that the policies cannot contain inadmissible cycles, as shown in Example 4.

Example 4. *Consider a scenario where policy(S, "Token length for password reset") is*

(50,	∅)
(30,	{("*Token duration (hours) for password reset*", 3)})

and policy(S, "Token duration (hours) for password reset") is

(3,	{("*Token length for password reset*", 40)})

By quickly inspecting the graphical representation in Fig. 2, we can easily spot the fact that the policy must be considered inadmissible – it contains a cycle such that when "Token duration (hours) for password reset" has a value of 30, a minimum value of 3 would be required for "Token duration (hours) for password reset", but the latter would in turn require a minimum value of 40 for "Token duration (hours) for password reset".

The policy would indeed violate Requirement R4 with $A_1 =$ *"Token length for password reset",* $A_2 =$ *"Token duration (hours) for password reset",* $w_1 = 30$, $w_2 = 3$, *and* $\overline{w} = 40$.

Requirement R5 ensures that the policies cannot contain useless "short circuits", as shown in Example 5.

Example 5. *Consider a scenario where policy(S, "Token length for password reset") is*

[1] Observe that we introduce the notion of expanded policy only to make the formalism in the remainder more compact – expanded policies do not need to be actually computed when implementing the proposed framework.

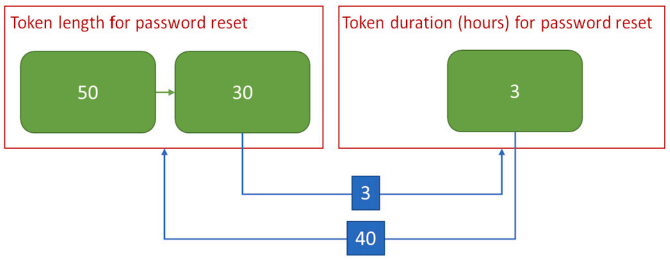
Token length for password reset
50
30
Token duration (hours) for password reset
3
3
40

Fig. 2. Example system policy with an inadmissible cycle.

$(50, \quad \emptyset)$
$(30, \quad \{(\text{“}\textit{Single use token for password reset”}, \textit{true}), (\text{“}\textit{Token duration (hours) for password reset”}, 3)\}\ )$

and policy$(S,$*“Single use token for password reset”*$)$ *is*

$(\textit{true}, \quad \{(\text{“}\textit{Token duration (hours) for password reset”}, 5)\}\ )$

As Fig. 3 shows, the policy must be considered inadmissible because:

- *when “Token length for password reset” has a value of* 30*, the policy would require “Token duration (hours) for password reset” to be at least* 3 *and “Single use token for password reset” to be* true*;*
- *the fact that “Single use token for password reset” is* true *in turn requires “Token duration (hours) for password reset” to be at least* 5*;*
- *thus, requiring “Token duration (hours) for password reset” to be at least* 3 *would be a “short-circuit” not actually applied.*

The policy would indeed violate Requirement R5 with $A_1 =$*“Token length for password reset”,* $A_2 =$*“Single use token for password reset”,* $A_3 =$*“Token duration (hours) for password reset”,* $w_1 = 30$*,* $w_2 = \textit{true}$*,* $w_3 = 5$*, and* $\overline{w} = 3$.

Requirement R6 ensures that (i) no attributes are used that are voided by the values of other attributes and (ii) if an attribute appears in C_p, then it must appear in all C_i's with $i > p$ with a higher minimum value (unless it is voided). The former case is shown in Example 6. The latter case is a “common-sense” consequence of Requirement R1 which imposes $m_i \preceq m_p$.

Example 6. *Consider a scenario where “PIN length for password reset”* $\in$ *voided-by(“Method for password reset”, URL token) and policy*$(S,$*“Method for password reset”*$)$ *is*

$(\textit{Offline methods}, \quad \emptyset)$
$(\textit{PINs}, \quad \{(\text{“}\textit{PIN length for password reset”}, 6)\}\ )$
$(\textit{URL token}, \quad \{(\text{“}\textit{PIN length for password reset”}, 6), (\text{“}\textit{Single use token for password reset”}, \textit{true})\}\ )$

The policy must be considered inadmissible because C_3 *fixes a minimum value for “PIN length for password reset” even if this attribute is voided by the value used for “Method for password reset” (URL token). The policy would violate Requirement R6 (first bullet) with* $A' =$*“PIN length for password reset”,* $v = 6$*,* $i = 3$*, and “PIN length for password reset”* $\in V$.

Consider now a scenario where:

- *“Method for password reset”* $\notin$ *voided-by(“Password length”,* 8*);*
- *“PIN length for password reset”* $\notin$ *voided-by(“Password length”,* 8*);*
- *“PIN length for password reset”* $\in$ *voided-by(“Method for password reset”, URL token);*

and policy$(S,$*“Password length”*$)$ *is*

$(16, \quad \emptyset)$
$(8, \quad \{(\text{“}\textit{Method for password reset”}, \textit{URL token}), (\text{“}\textit{PIN length for password reset”}, 6)\}\ )$

Again, the policy must be considered inadmissible because C_2 *fixes a minimum value for “PIN length for password reset” even if this attribute is voided by the value used for “Method for password reset” (URL token). However, differently from the previous scenario, the latter value is used in* C_2 *itself. The policy would violate Requirement R6 (first bullet) with* $A' =$*“PIN length for password reset”,* $v = 6$*,* $i = 2$*, and “PIN length for password reset”* $\in V$.

3. User registration

Suppose $U \in \mathcal{U}$ wants to create an account on system $S \in \mathcal{S}$. For each $A \in \mathcal{A}_{usr}$, we write

$\textit{usr-val}(S, A, U) \in \textit{values}(A)$

to denote the value of attribute A that user U wants to use for their registration.

To make the notation more compact, in the following we use a single *verification value* function for both system and attribute values. In particular, we define $\textit{ver-val}(S, A, U)$ as $\textit{sys-val}(S, A)$ if $A \in \mathcal{A}_{sys}$, and $\textit{usr-val}(S, A, U)$ if $A \in \mathcal{A}_{usr}$. Observe that, in the former case, the user is not taken into account — this is equivalent to assuming a single value for all users on system attributes.

System S will perform the registration of U only if, $\forall A \in \mathcal{A}_{auth}$, $\exists (m_i, C_i) \in \textit{policy}(S, A)$ such that

- $\textit{ver-val}(S, A, U) \succeq m_i$[2];
- $\forall (A', v) \in C_i$, $\textit{ver-val}(S, A', U) \succeq v$.

Intuitively, the registration will be verified iff, for each authentication attribute A, we can find a pair (m_i, C_i) in the policy for A such that the value of A is at least m_i and all attributes in C_i have a value that is higher than or equal to their associated minimum accepted value.

Example 7. *Consider a system* S *with policy*$(S,$*“Password length”*$)$ *consisting of the following pairs:*

$(20, \quad \emptyset\ )$
$(8, \quad \{(\text{“}\textit{Method for password reset”}, \textit{PINs})\}\ )$

In this case, for a given user U*, we could have:*

- $\textit{usr-val}(S,$*“Password length”*$, U) = 20$ *and* $\textit{usr-val}(S,$*“Method for password reset”*$, U) =$ *Security questions. In this case, the registration of* U *can be performed because* $(20, \emptyset) \in \textit{policy}(S,$*“Password length”*$)$ *and* $\textit{ver-val}(S,$*“Password length”*$, U) \succeq 20$.
- $\textit{usr-val}(S,$*“Password length”*$, U) = 10$ *and* $\textit{usr-val}(S,$*“Method for password reset”*$, U) =$ *Offline methods. In this case, the registration of* U *can be performed because* $(8, \{(\text{“}\textit{Method for password reset”}, \textit{PINs})\}) \in \textit{policy}(S,$*“Password length”*$)$*,* $\textit{ver-val}(S,$*“Password length”*$, U) \succeq 8$*, and* $\textit{ver-val}(S,$*“Method for password reset”*$, U) \succeq$ *PINs.*

4. Modeling and verifying interactions

In this section we provide our formalization of the interactions among authentication systems in the form of dependencies. Then, we develop an algorithm to verify whether such dependencies can be established given the policies of the involved systems.

[2] The condition can be equivalently written as $\textit{ver-val}(S, A, U) = m_i$ if we use $\textit{exp-policy}(S, A)$ instead of $\textit{policy}(S, A)$.

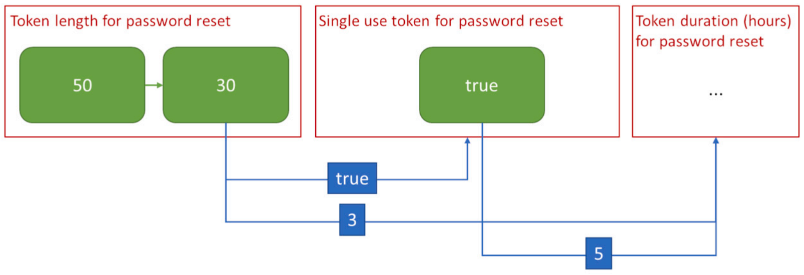
Token length for password reset
50
30
Single use token for password reset
true
Token duration (hours) for password reset
...
true
3
5

Fig. 3. Example system policy with a useless "short-circuit".

4.1. Dependencies

We model interactions among authentication systems by means of a *dependency relation* "$\hookrightarrow$". In particular, given two systems $S, S' \in \mathcal{S}$ and a user $U \in \mathcal{U}$, we write

$$S \xrightarrow{U} S'$$

to denote the fact that system S *depends on system* S' *for the authentication of user* U. In the following, we discuss two main kinds of dependencies that may arise in real world scenarios.

Multiple dependencies Consider two dependencies $S \xrightarrow{U} S'$ and $S \xrightarrow{U} S''$, that is system S depends on both systems S' and S'' for the authentication of user U. These models, e.g., those scenarios where authentication relies on other (sub)systems.

Example 8. *A few years ago, Apple's iCloud system allowed authentication through various interfaces (file repo, find-my-iPhone, app data sync, etc.). However, it did not enforce the same security policy at all interfaces (in particular, it did not throttle password attempts at the find-my-iPhone interface). In our framework, after fixing policy(iCloud, "Throttling applied to passwords") = $\{(true, \emptyset)\}$, we expect any system S' with sys-val(S', "Throttling applied to passwords") = $false$ to be ineligible for a dependency iCloud$\xrightarrow{U} S'$.*

Derived dependencies Consider two dependencies $S \xrightarrow{U} S'$ and $S' \xrightarrow{U} S''$, that is system S depends on system S' which in turn depends on system S'' for the authentication of user U. This implies a dependency of system S on system S'', which we call a *derived dependency* — such dependencies must be verified as well.

Example 9. *A few years ago, a famous gmail account was hacked using a password reset performed though a link sent to a recovery email @me.com (managed by Apple). The latter allowed resetting the password using a billing address and the last 4 digits of a credit card number. The user had an Amazon account, and Amazon's account management system did not require login to purchase — in addition, users could add credit cards to other users' accounts. Finally, resetting a password in Amazon required one of the credit card numbers, and after login, the website showed the last 4 digits of all the registered cards. In our framework, we would need to establish both the dependencies $gmail \xrightarrow{U} me.com$ (since gmail relies on me.com for password recovery) and $me.com \xrightarrow{U} Amazon$ (since the user shares the same secret — the last 4 digits of a credit card number — with the two systems). However, after fixing policy(gmail, "Method for password reset") = $\{(a, \emptyset)\}$ with any $a \succ$Security questions, the dependency $gmail \xrightarrow{U} me.com$ could not be established.*

In the following, we use $S \overset{U}{\rightsquigarrow} S''$ to denote both direct and derived dependencies, i.e.,

$$S \overset{U}{\rightsquigarrow} S'' \text{ if } S \xrightarrow{U} S'' \text{ or } S \xrightarrow{U} S' \text{ and } S' \overset{U}{\rightsquigarrow} S''.$$

4.2. Dependency verification

In our framework, Algorithm Verify-Dependency is invoked before establishing every new dependency $S \xrightarrow{U} S'$. The algorithm verifies whether all the required conditions are met. The process start with $(S, S', U, policy, ver\text{-}val, \emptyset)$ as input.

Algorithm 1 Verify-Dependency.

```
Input: Systems $S, S' \in \mathcal{S}$, user $U \in \mathcal{U}$, functions policy and ver-val, set $\overline{S} \subseteq \mathcal{S}$ of systems
Output: true if $S \xrightarrow{U} S'$ can be established, false otherwise
 1: if $S \overset{U}{\rightsquigarrow} S'$ then
 2:     return true
 3: $\overline{S}.add(S)$
 4: for all $S'' \in \mathcal{S}$ s.t. $S'' \xrightarrow{U} S$ and $S'' \notin \overline{S} \cup \{S'\}$ do
 5:     if not Verify-Dependency($S'', S', U, policy, ver\text{-}val, \overline{S}$) then
 6:         return false
 7: $\hat{S} \leftarrow \{S\}$
 8: for all $S'' \in \mathcal{S}$ s.t. $S \overset{U}{\rightsquigarrow} S''$ do
 9:     $\hat{S}.add(S'')$
10: if not Verify-Forward($S, S', U, policy, ver\text{-}val, \hat{S}$) then
11:     return false
12: return true
```

Algorithm 2 Verify-Forward.

```
Input: Systems $S, S' \in \mathcal{S}$, user $U \in \mathcal{U}$, functions policy and ver-val, set $\overline{S} \subseteq \mathcal{S}$ of systems
Output: true if $S \xrightarrow{U} S'$ can be established, false otherwise
 1: for all $A \in \mathcal{A}_{auth}$ do
 2:     $val \leftarrow ver\text{-}val(S', A, U)$
 3:     verified ← false
 4:     $\{(m_1, C_1), \ldots, (m_n, C_n)\} \leftarrow policy(S, A)$
 5:     for $i$ in $1, \ldots, n$ do
 6:         if $val \geq m_i$ and $\forall (A', v) \in C_i$, $ver\text{-}val(S', A', U) \geq v$ then
 7:             verified ← true // can also break the loop started on Line 5
 8:     if not verified then
 9:         return false
10: $\overline{S}$.add($S'$)
11: for all $S'' \in \mathcal{S}$ s.t. $S' \xrightarrow{U} S''$ and $S'' \notin \overline{S}$ do
12:     if not Verify-Forward($S, S'', U, policy, ver\text{-}val, \overline{S}$) then
13:         return false
14: return true
```

The algorithm immediately returns *true* if a dependency of S on S' already exists (Lines 1–2). Lines 3–6 traverse the dependencies backwards, with recursive invocations that verify whether the systems that depend on S are allowed to depend on S' as well. Finally (Lines 7–11), the algorithm builds the set $\hat{S}$ of systems that already depend on S (i.e., before adding $S \xrightarrow{U} S'$) and invokes Algorithm Verify-Forward to verify whether the dependency can be established, considering such systems as already verified.

In Algorithm VERIFY-FORWARD, Lines 1–9 verify whether system S' and user U satisfy system S's policy for all authentication attributes. In particular, the loop starting on Line 5 tries to find a pair (m_i, C_i) such that the verification value of A under S', along with the verification values of all attributes in C_i under S', satisfy the policy requirements stated by S.[3] If the verification succeeds, then S' is added to the set $\overline{S}$ of verified systems (Line 10). Finally, for all systems that depend on S' and have not been verified (Line 11), the algorithm invokes itself recursively.

4.3. Towards strong dependencies

A possible alternative notion of dependency could just look at policies instead of considering the actual values used for authentication attributes.

We start by introducing a partial order on policies. Given two systems $S, S' \in \mathcal{S}$ and an attribute $A \in \mathcal{A}_{auth}$, we write

$$policy(S, A) \leq_P policy(S', A)$$

iff $\forall (m_j, C_j) \in policy(S', A), \exists (m_i, C_i) \in policy(S, A)$ such that:

- $m_i \preceq m_j$;
- $\forall (A', v) \in C_i,\ \exists (A', v') \in C_j$ such that $v \preceq v'$.

In other words, every pair (m_j, C_j) in $policy(S', A)$ is "stronger" than at least one pair (m_i, C_i) in $policy(S, A)$. In order to consider (m_j, C_j) stronger than (m_i, C_i), we obviously require that $m_i \preceq m_j$ — but in addition, each attribute that is mentioned in C_i must be mentioned in C_j as well, with a greater or equal associated minimum value.

With the above definition at hand, it is easy to see that if a user satisfies the policy of S' (i.e., they can perform the registration under S'), then they also satisfy the policy of S. We can then introduce the concept of strong dependency. Specifically, a *strong dependency of S on S' for the authentication of user U*, denoted $S \overset{U}{\hookrightarrow} S'$, can be established iff $\forall A \in \mathcal{A}_{auth}$, $policy(S, A) \leq_P policy(S', A)$.

The main advantage of strong dependencies is that they are transitive by definition, so $S \overset{U}{\hookrightarrow} S'$ and $S' \overset{U}{\hookrightarrow} S''$ *implies* $S \overset{U}{\hookrightarrow} S''$. Thus, there is no need to verify derived dependencies — when a new dependency is established between two systems, *no interaction with other systems is needed.* On the other hand, strong dependencies could end up being too "conservative" and less capable of modeling real world use cases. One possible option to overcome this drawback could be that of introducing an (also partial) order among the authentication attributes in order to derive a less restrictive definition of order on policies. We plan to investigate this in future work.

5. Security analysis

In this section, we provide a formal analysis of the security of authentication in our proposed framework.

We start by introducing the concept of capability of an adversary. Given an adversary **Adv** and an attribute A, we define the *capability* of **Adv** on A as $cap(\mathbf{Adv}, A) \in values(A)$. Intuitively, $cap(\mathbf{Adv}, A)$ is the maximum value of A that **Adv** is able to compromise. For instance, consider $A =$"Password length" with $cap(\mathbf{Adv}, A) = 10$. This means that **Adv** is able to break users' passwords of length at most 10.

Observe that, in principle, the actual capability of an adversary may depend on the value of other attributes. For example, **Adv** may not be able to bypass a throttling mechanism ($cap(\mathbf{Adv},$"Throttling applied to password"$) = false$), so when throttling is applied, the actual length of the passwords that **Adv** can break is lower than 10. *We make the worst case assumption that the capability on an attribute is independent of the values assumed by other attributes.*

We now give our definition of vulnerable policy. Given a system $S \in \mathcal{S}$ and an attribute A, we say that $policy(S, A)$ is *vulnerable* if there exists an adversary **Adv** and a pair $(m_i, C_i) \in policy(S, A)$ such that:

- $cap(\mathbf{Adv}, A) \succeq m_i$
- $\forall (A', v) \in C_i,\ cap(\mathbf{Adv}, A') \succeq v$.

In other words, $policy(S, A)$ is vulnerable if it contains a pair (m_i, C_i) and an adversary is able to compromise attribute A with value m_i together with all the attributes in C_i.

Example 10. *Assume $policy(S,$"Password length"$) = \{(10, \emptyset), (8, \{($"Throttling applied to password"$, true)\})\}$. Suppose there exists an adversary **Adv** with $cap(\mathbf{Adv},$"Password length"$) = 10$ and $cap(\mathbf{Adv},$"Throttling applied to password"$) = false$. In this case, the policy is vulnerable based on the first pair. Observe that, this may represent an issue for S — indeed, even if the system applies a throttling mechanism, it may establish a dependency with a system S' which does not throttle, and thus be potentially vulnerable.*

In order to take into account the values actually used for attributes, we define function *break*. Given a system S, an attribute A, a set of attributes $\mathcal{A}^* \subset \mathcal{A}_{auth}$, and a user $U \in \mathcal{U}$, $break(S, A, \mathcal{A}^*, U)$ is *true* iff $\exists \mathbf{Adv}$ such that $cap(\mathbf{Adv}, A) \succeq ver\text{-}val(S, A, U)$ and $\forall A^* \in \mathcal{A}^*, cap(\mathbf{Adv}, A^*) \succeq ver\text{-}val(S, A^*, U)$. Observe that the function works on a set of attributes $\mathcal{A}^*$ — in the following, they will correspond to the attributes in a pair in $policy(S, A)$.

We leverage function *break* to define the concept of directly vulnerable system. We say that a system $S \in \mathcal{S}$ is *directly vulnerable* on an attribute $A \in \mathcal{A}_{auth}$ for a user $U \in \mathcal{U}$ if

$$\forall (m_i, C_i) \in policy(S, A),\ break(S, A, \mathcal{A}^*, U) = true$$

where $\mathcal{A}^* = \{A^* \in \mathcal{A}_{auth} : (A^*, v) \in C_i\}$. Observe that:

- The notion of direct vulnerability does not take into account the system's dependencies — the latter will be considered in the definition of indirect vulnerability.
- The definition looks at all pairs $(m_i, C_i) \in policy(S, A)$. This is due to the fact that if the attacker is not capable of breaking the pair(s) (m_i, C_i) used during user registration, then S is secure on A for U.

Theorem 1 states that, as long as the policy set by a system on a given attribute is sufficiently strong, the user registration procedure prevents an attacker from breaking the system for that user.

Theorem 1. *Consider a system S, an attribute A, and a user U performing user registration as defined in Section 3. If $policy(S, A)$ is not vulnerable, then S is not directly vulnerable on A for U.*

Proof. By definition, S is not directly vulnerable on A for U if $\exists (m_i, C_i) \in policy(S, A)$ such that $break(S, A, \mathcal{A}^*, U) = false$ where $\mathcal{A}^* = \{A^* \in \mathcal{A}_{auth} : (A^*, v) \in C_i\}$. After the user registration process, we have $\exists (\overline{m}, \overline{C}) \in policy(S, A)$ such that $ver\text{-}val(S, A, U) \succeq \overline{m}$ and $\forall (A', v) \in \overline{C}, ver\text{-}val(S, A', U) \succeq v$. By definition, if $policy(S, A)$ is not vulnerable, then $\forall (m, C)$ and for any adversary **Adv**, $cap(\mathbf{Adv}, A) \prec m$ and $\forall (A', v) \in C, cap(\mathbf{Adv}, A') \prec v$. Then, for any adversary **Adv**, it holds that $ver\text{-}val(S, A, U) \succ cap(\mathbf{Adv}, A)$ and $\forall (A', v) \in \overline{C}$, $ver\text{-}val(S, A', U) \succ cap(\mathbf{Adv}, A')$. This corresponds to $break(S, A, \mathcal{A}^*, U) = false$ where $\mathcal{A}^* = \{A^* \in \mathcal{A}_{auth} : (A^*, v^*) \in \overline{C}\}$. □

We now introduce the notion of indirect vulnerability, which may happen when a system can be compromised through other systems on which it depends. We say that a system $S \in \mathcal{S}$ is *indirectly vulnerable*

[3] The first condition on Line 6 can be equivalently written as $val = v_i$ instead of $val \succeq v_i$ if we use $exp\text{-}policy(S, A)$ instead of $policy(S, A)$.

on an attribute $A \in \mathcal{A}_{auth}$ for a user $U \in \mathcal{U}$ if $\exists S_k \in \mathcal{S} \setminus \{S\}$ with $S \overset{U}{\rightsquigarrow} S_k$ such that $\forall (m_i, C_i) \in policy(S, A), break(S_k, A, \mathcal{A}^*, U) = true$ where $\mathcal{A}^* = \{A^* \in \mathcal{A}_{auth} : (A^*, v) \in C_i\}$.

Theorem 2 states that, as long as the policy set by a system on a given attribute is sufficiently strong, the system cannot be compromised through other systems on which it depends.

Theorem 2. *Consider a system S, a user U, a system $S_k \in \mathcal{S} \setminus \{S\}$ such that $S \overset{U}{\rightsquigarrow} S_k$, and an attribute A. If policy(S, A) is not vulnerable, then S is not indirectly vulnerable on A for U.*

Proof. Since by hypothesis $S \overset{U}{\rightsquigarrow} S_k$, S is not indirectly vulnerable on A for U if $\exists (m_i, C_i) \in policy(S, A)$ such that $break(S_k, A, \mathcal{A}^*, U) = false$ where $\mathcal{A}^* = \{A^* \in \mathcal{A}_{auth} : (A^*, v) \in C_i\}$. The fact that $S \overset{U}{\rightsquigarrow} S_k$ implies that Algorithm VERIFY-FORWARD returns *true* when invoked with S and S_k as input. Therefore, $\exists (\overline{m}, \overline{C}) \in policy(S, A)$ such that $ver\text{-}val(S_k, A, U) \geq \overline{m}$ and $\forall (A', v) \in \overline{C}, ver\text{-}val(S_k, A', U) \geq v$. By definition, if $policy(S, A)$ is not vulnerable then $\forall (m, C)$ and for any adversary **Adv**, $cap(\mathbf{Adv}, A) \prec m$ and $\forall (A', v) \in C, cap(\mathbf{Adv}, A') \prec v$. Then, for any adversary **Adv**, it holds that $ver\text{-}val(S_k, A, U) \succ cap(\mathbf{Adv}, A)$ and $\forall (A', v) \in \overline{C}$, $ver\text{-}val(S_k, A', U) \succ cap(\mathbf{Adv}, A')$. This corresponds to $break(S_k, A, \mathcal{A}^*, U) = false$ where $\mathcal{A}^* = \{A^* \in \mathcal{A}_{auth} : (A^*, v^*) \in \overline{C}\}$. □

6. Implementation of the framework

In this section we define a reference set of authentication attributes that could be used when adopting our proposed framework. Then, after providing some background notions on the Self-Sovereign Identity (SSI) paradigm, we propose an SSI-based implementation of our framework. Finally, we discuss a possible extension to encompass identification attributes.

6.1. Reference authentication attributes

Tables 1 and 2 report a possible reference set of authentication attributes derived from the *OWASP authentication cheat sheet* (OWASP, 2024) and *NIST Digital Identity Guidelines: Authentication and Lifecycle Management* (NIST, 2017a). The set is not meant to be exhaustive, but just to show how our approach can be applied in real world scenarios. We selected 6 user attributes and 26 system attributes. Below, we provide additional details on some of the attributes.

A_1 denotes the means through which the user authenticates. It is considered a user attribute because systems may offer multiple types of authenticators and the user can choose which one to use. NIST (2017a) identifies 9 types of authenticators that can be used in combination: Password, Look-Up Secret, Out-of-Band Device, Single-Factor (SF) One-time Password (OTP) Device, SF Crypto Software, SF Crypto Device, Multi-Factor (MF) OTP Device, MF Crypto Software, and MF Crypto Device. We only considered the combinations reported in (NIST, 2017a) – other combinations are possible. The order of *values*(A_1) derives from the following reasoning. First, a single-factor authenticator is less secure than a multi-factor authenticator of the same type. Similarly, we consider a Crypto Software authenticator less secure than a Crypto Device authenticator. Moreover, NIST (2017a) defines the three Authenticator Assurance Levels (AAL) reported below.

- At AAL1, any of the 9 authenticator types above can be adopted.
- At AAL2, the following authenticator types can be adopted: MF OTP Device, MF Crypto Software, MF Crypto Device, Password plus Look-Up Secret, Password plus Out-of-Band, Password plus SF OTP Device, Password plus SF Crypto Software, and Password plus SF Crypto Device.
- At AAL3, the following authenticator types can be adopted: MF Crypto Device, Password plus SF Crypto Device, SF OTP Device plus SF Crypto Software plus Password, MF OTP Device plus SF Crypto Software, SF OTP Device plus MF Crypto Software, and MF OTP Device plus SF Crypto Device.

Based on the above AALs, a partial order can be derived. For example, since the MF Crypto Device authenticator is allowed by all of the three AALs, it can be considered more (or at least equally) secure than all other authenticator types. Similarly, since the MF OTP Device authenticator is allowed by AAL2, it can be considered more secure than the Look-Up Secret authenticator allowed (when not in combination with other authenticators) by AAL1. However, to obtain a total order, some assumptions were made also based on qualitative reasoning. For example, we considered the use of an out-of-band device more secure than a Look-Up Secret since it relies on an external communication channel. Under this assumption, a Password plus Look-Up Secret authenticator precedes a Password plus Out-of-Band authenticator in the total order. By applying a similar reasoning, we obtained the values for A_1 ordered as reported in Table 1.

A_6 models the cases where the password is chosen by the subscriber (as in the most common authentication methods) or by the verifier. Example 11 shows a use of the attribute.

Example 11. *NIST Special Publication 800-63B (NIST, 2017a) recommends the following:*

> *"Memorized secrets SHALL be at least 8 characters in length if chosen by the subscriber. Memorized secrets chosen randomly by the CSP or verifier SHALL be at least 6 characters in length and MAY be entirely numeric."*

We can express the above policy by using the attributes in Table 1 and making policy(S, A_6) consist of the following pairs:

$$\begin{array}{rl} (\textit{verifier}, & \{(A_2, 6)\}\ ) \\ (\textit{subscriber}, & \{(A_2, 8), \\ & (A_3, \textit{true})\}\ ) \end{array}$$

For A_9, we assume that each system maintains a dictionary and checks whether the password contains words from the dictionary. We assume it is not mandatory for the system to reject a password in this case (it could simply show a warning to the user).

It should also be observed that attributes such as A_{12} and A_{22} appear similar, and this needed for exact compliance with the notation introduced in Section 2.2. Indeed, a single "Salt length" attribute would not suffice for two main reasons:

- According to our definition of policy, "Salt length" may be used without specifying the authenticator type in A_1. However, with a single "Salt length" attribute, it would not be clear whether it refers to the salt for the password or for the look-up secret.
- When the value of A_1 is Password plus Look-Up Secret, two different values for "Salt length" should be specified.

The same considerations apply, e.g., to attributes A_{13} and A_{23} – for brevity, we do not report all the attributes with such properties.

To conclude this section, we report in Tables 3 and 4, for each attribute A in Tables 1 and 2, the corresponding set of voided attributes for each possible value of A. For example, when the value of attribute A_1 is Password, the attributes A_{19}–A_{32} cannot be used as they are not related to password-based authentication. When more authenticator types are used in combination, the voided attributes are obtained as the intersection of the voided attributes of the single authenticator types. In the tables, "(Any)" denotes any attribute in *values*(A) – in other words, the set of voided attributes is the same regardless of the value of A. For example, for any value of A_2 ("Password Length"), attribute A_{30}

Table 1
Reference authentication attributes (1/2).

Attribute A	Name	$values(A)$
$A_1 \in \mathcal{A}_{usr}$	"Permitted authenticator type"	⟨Password, Look-Up Secret, Out-of-Band, SF OTP Device, SF Crypto Software, SF Crypto Device, Password plus Look-Up Secret, Password plus Out-of-Band, Password plus SF OTP Device, Password plus SF Crypto Software, MF OTP Device, MF Crypto Software, Password plus SF Crypto Device, SF OTP Device plus SF Crypto Software plus Password, MF OTP Device plus SF Crypto Software, SF OTP Device plus MF Crypto Software, MF OTP Device plus SF Crypto Device, MF Crypto Device⟩
$A_2 \in \mathcal{A}_{usr}$	"Password length"	$\langle 4, \dots, 256 \rangle$
$A_3 \in \mathcal{A}_{sys}$	"No password character limit"	⟨false, true⟩
$A_4 \in \mathcal{A}_{sys}$	"No password composition rules"	⟨false, true⟩
$A_5 \in \mathcal{A}_{sys}$	"Transmit password over TLS"	⟨false, true⟩
$A_6 \in \mathcal{A}_{sys}$	"Password chooser"	⟨subscriber, verifier⟩
$A_7 \in \mathcal{A}_{sys}$	"Explicit limited password length"	⟨false, true⟩
$A_8 \in \mathcal{A}_{sys}$	"No silent password truncation"	⟨false, true⟩
$A_9 \in \mathcal{A}_{usr}$	"No dictionary words used for passwords"	⟨false, true⟩
$A_{10} \in \mathcal{A}_{sys}$	"Throttling applied to passwords"	⟨false, true⟩
$A_{11} \in \mathcal{A}_{sys}$	"Method for storing password"	⟨cleartext, one-way function, salted one-way function, one-way key derivation function⟩
$A_{12} \in \mathcal{A}_{sys}$	"Salt length for storing password"	$\langle 10, \dots, 100 \rangle$
$A_{13} \in \mathcal{A}_{sys}$	"Cost count key derivation function for storing password"	$\langle 10000, \dots, 100000 \rangle$
$A_{14} \in \mathcal{A}_{usr}$	"Method for password reset"	⟨Security questions, URL token, PINs, Offline methods⟩
$A_{15} \in \mathcal{A}_{sys}$	"Token length for password reset"	$\langle 1, \dots, 1000 \rangle$
$A_{16} \in \mathcal{A}_{sys}$	"Single use token for password reset"	⟨false, true⟩

("Second factor protection for MF OTP Device") cannot be used. This is the only voided attribute because A_{30} is the only attribute that refers to a single authenticator type (MF OTP Device) that cannot be used in combination with the password.

6.2. Background notions on self-sovereign identity

Self-Sovereign Identity is a decentralized paradigm for managing digital identities. In real-world implementations, it is based on blockchain technology (Mühle et al., 2018). In this section, we discuss the most relevant features of each building block of SSI used in our framework.

Verifiable credentials A *Verifiable Credential* (VC) is a set of claims about a subject representing information issued by a certain authority. Such authority is called the *issuer* while the subject of the VC is called the *holder*. At a high level, the VC includes three sections:

- VC's metadata (e.g., date of issue, expiration date, state of the credential, and so on);
- VC's claims about the subject;
- Proof, i.e., the digital signature made by the issuer of the credential.

The VCs are stored by the holder in a *digital wallet*. The holder can prove something about themselves by presenting a *Verifiable Presentation* (VP) to a verifier that consists of data derived from one or more VCs. The verifier can define an access control policy to provide a given service to the holder whose VPs satisfy its policy.

VCs often coincide with VPs. However, for privacy reasons, VPs may disclose less information than VCs. For example, the selective disclosure (Mukta et al., 2020) and ZKP proofs (Preukschat and Reed, 2021) mechanisms can be adopted. With selective disclosure, the holder can make fine-grained decisions about what information (among all the claims in the VCs) to share with the verifier. ZKP proofs consist in cryptographic methods that the holder can adopt to prove to the verifier that they know a certain value without actually disclosing the value.

Verification process After the presentation, the verifier performs the verification process through the following steps:

- VC's state verification: the verifier checks that the VC is still valid, i.e., it has not been revoked.
- VC's claims verification: the verifier checks that the information contained in the VC satisfies the access control policy. In our proposal, this step is performed by Algorithm 1.
- VC's issuer verification: the verifier checks the authenticity of the VC's issuer. The VC contains the *Decentralized Identifier* (DID) of the issuer. Based on this value, the verifier accesses the blockchain to

Table 2
Reference authentication attributes (2/2).

Attribute A	Name	$values(A)$
$A_{17} \in \mathcal{A}_{sys}$	"Token duration (hours) for password reset"	$\langle 24, \ldots, 1 \rangle$
$A_{18} \in \mathcal{A}_{sys}$	"PIN length for password reset"	$\langle 4, \ldots, 12 \rangle$
$A_{19} \in \mathcal{A}_{sys}$	"Bits of entropy for look-up secret"	$\langle 20, \ldots, 200 \rangle$
$A_{20} \in \mathcal{A}_{sys}$	"Throttling applied to look-up secrets"	⟨false, true⟩
$A_{21} \in \mathcal{A}_{sys}$	"Method for storing look-up secret"	⟨cleartext, one-way function, salted one-way function, one-way key derivation function⟩
$A_{22} \in \mathcal{A}_{sys}$	"Salt length for storing look-up secret"	$\langle 10, \ldots, 100 \rangle$
$A_{23} \in \mathcal{A}_{sys}$	"Cost count key derivation function for storing look-up secret"	$\langle 10000, \ldots, 100000 \rangle$
$A_{24} \in \mathcal{A}_{usr}$	"Out-of-band device protection"	⟨no protection, biometric, PIN, passcode⟩
$A_{25} \in \mathcal{A}_{sys}$	"Bits of entropy of out-of-band secrets"	$\langle 20, \ldots, 200 \rangle$
$A_{26} \in \mathcal{A}_{sys}$	"Maximum Authentication time (minutes) for out-of-band device"	$\langle 100, \ldots, 10 \rangle$
$A_{27} \in \mathcal{A}_{sys}$	"Secret key length for OTP devices"	$\langle 64, \ldots, 1024 \rangle$
$A_{28} \in \mathcal{A}_{sys}$	"Nonce length for OTP devices"	$\langle 4, \ldots, 16 \rangle$
$A_{29} \in \mathcal{A}_{sys}$	"Nonce expiration (minutes) for OTP devices"	$\langle 10, \ldots, 1 \rangle$
$A_{30} \in \mathcal{A}_{usr}$	"Second factor protection for MF OTP devices"	⟨biometric, passcode⟩
$A_{31} \in \mathcal{A}_{sys}$	"Key storage for Crypto Software"	⟨TEE, TPM, HSM⟩
$A_{32} \in \mathcal{A}_{sys}$	"Phisical input for Crypto Device"	⟨false, true⟩

obtain the public key of the issuer, so they can verify the proof of the VC.

Credential revocation The issuer can revoke the VC by leveraging a cryptographic accumulator, stored in the blockchain, that allows the accumulation of elements from a finite set into a concise value by means of a one-way hash function. When the verifier performs the verification of the VCs provided by the holder it should check that they are not revoked by testing their presence in the cryptographic accumulator.

6.3. SSI-based implementation

Our SSI-based implementation is based on the Trinsic Ecosystems (Trinsic, 2024), implementing the Sovrin Framework (Tobin and Reed, 2016). This is a reference technology adhering to the main SSI standards in terms of format of the verifiable credentials (compliant with W3C specifications (W3C, 2022a,b)) and ToIP stack (Davie et al., 2019). An advantage of this SSI implementation is its capability to support ZKP proofs and selective disclosure mechanisms. However, *our solution does not strongly rely on a specific SSI implementation*, but different implementations of the SSI framework can be adopted.

We recall that $\mathcal{S}$ represents the set of systems, $\mathcal{U}$ the set of users, and $\mathcal{A}_{auth} = \mathcal{A}_{usr} \cup \mathcal{A}_{sys}$ the set of authentication attributes.

System attributes certification Suppose the existence of trusted authorities in charge of certifying the system attributes for each system $S \in \mathcal{S}$.

In our implementation, system S contacts a trusted authority to obtain a verifiable credential VC_S containing the values of its system attributes. According to the W3C specification, VC_S can be expressed in JSON-LD format and includes (in the claim part), for each $A \in \mathcal{A}_{sys}$, a pair $\{name : value\}$ where $name = A$ and $value = sys\text{-}val(S, A)$. Observe that, according to the SSI terminology, the trusted authority is the issuer of VC_S while S is the holder.

After receiving VC_S, S makes it publicly available through an endpoint (reachable, e.g., through HTTPS). The endpoint will be accessed by other systems during the dependency verification phase (see below) to verify compliance with their policy.

An aspect that deserves further consideration is the role of the trusted authorities that release the verifiable credentials. These actors have a relevant role in our protocol, but in real-life applications of the protocol, their presence could be considered implausible at a first glance. An important question is therefore: *How to transfer the conceptual role of these authorities into a real-life implementation of our solution?* To answer this, we start by noting that the verifiable credentials are in fact self-signed by the authentication systems and, if the latter adhere to the whole system, they are subject to audits and checks carried out by independent bodies (i.e., trusted parties). As a matter of fact, the trend outlined by EU regulation (eIDAS, 2014) is to ground the security of digital identity systems on the presence of trusted services. In addition, checks do not always require specific inspections, but can also be performed in a black-box fashion (as for penetration testing) for most of the system attributes. For instance, the fact that a system implements a throttling mechanism or not can be detected by measuring the time elapsed between consecutive failed authentication attempts. Thus, a system wrongly declaring to implement a throttling mechanism can be easily spotted and thus lose its reputation. On the other hand, black box evaluations are not helpful in some other cases, such as back-end attributes. In these cases, the combination of audits, (remote) inspections, and open source intelligence (to find traces that prove falsity of some claims about authentication attributes) can represent an effective way to prevent self promotion of untrustworthy authentication systems.

User attributes certification This procedure is similar to that described in the previous section. The main difference is that, in this case, the issuer's role is played by the system while the holder is represented by the user.

At the end of the user registration (and dependency verification) to a system $S \in \mathcal{S}$, a user $U \in \mathcal{U}$ receives a verifiable credential VC_U^S. The latter includes (in the claim part), for each $A \in \mathcal{A}_{usr}$, a pair $\{name : value\}$ where $name = A$ and $value = usr\text{-}val(S, A, U)$. Furthermore, a claim is stored in VC_U^S to identify the set of systems $\overline{S} = \{S' \in \mathcal{S} : S \overset{U}{\longrightarrow} S'\}$. In JSON format, the claim is a pair $\{name : value\}$ where $name =$"Dependent on" and $value$ is a list containing the DID of each

Table 3
Voided authentication attributes (1/2).

Attribute A	Value v	*voided-by*(A, v)
A_1	Password	$\{A_{19}, \dots, A_{32}\}$
	Look-Up Secret	$\{A_2, \dots, A_{18}, A_{24}, \dots, A_{32}\}$
	Out-of-Band	$\{A_2, \dots, A_{23}, A_{27}, \dots, A_{32}\}$
	SF OTP Device	$\{A_2, \dots, A_{26}, A_{30}, \dots, A_{32}\}$
	SF Crypto Software	$\{A_2, \dots, A_{26}, A_{32}\}$
	SF Crypto Device	$\{A_2, \dots, A_{31}\}$
	Password plus Look-Up Secret	*voided-by*$(A_1$, Password$)$ $\cap$ *voided-by*$(A_1$, Look-Up Secret$) = \{A_{24}, \dots, A_{32}\}$
	Password plus SF OTP Device	*voided-by*$(A_1$, Password$)$ $\cap$ *voided-by*$(A_1$, SF OTP Device$) = \{A_{19}, \dots, A_{26}, A_{30}, A_{32}\}$
	Password plus SF Crypto Software	*voided-by*$(A_1$,Password$)$ $\cap$ *voided-by*$(A_1$, SF Crypto Software$) = \{A_{19}, \dots, A_{30}, A_{32}\}$
	MF OTP Device	$\{A_2, \dots, A_{26}, A_{31}, A_{32}\}$
	MF Crypto Software	$\{A_2, \dots, A_{26}, A_{30}\}$
	Password plus SF Crypto Device	*voided-by*$(A_1$, Password$)$ $\cap$ *voided-by*$(A_1$, SF Crypto Device$) = \{A_{19}, \dots, A_{31}\}$
	SF OTP Device plus SF Crypto Software plus Password	*voided-by*$(A_1$, SF OTP Device$)$ $\cap$ *voided-by*$(A_1$, SF Crypto Software$)$ $\cap$ *voided-by*$(A_1$, Password$) = \{A_{19}, \dots, A_{30}\}$
	MF OTP Device plus SF Crypto Software	*voided-by*$(A_1$, MF OTP Device$)$ $\cap$ *voided-by*$(A_1$, SF Crypto Software$) = \{A_2, \dots, A_{26}, A_{32}\}$
	SF OTP Device plus MF Crypto Software	*voided-by*$(A_1$, SF OTP Device$)$ $\cap$ *voided-by*$(A_1$, MF Crypto Software$) = \{A_2, \dots, A_{26}, A_{30}\}$
	MF OTP Device plus SF Crypto Device	*voided-by*$(A_1$, MF OTP Device$)$ $\cap$ *voided-by*$(A_1$, SF Crypto Device$) = \{A_2, \dots, A_{26}, A_{31}\}$
	MF Crypto Device	$\{A_2, \dots, A_{31}\}$
A_2	(Any)	$\{A_{30}\}$
A_3	(Any)	$\{A_{30}\}$
A_4	(Any)	$\{A_{30}\}$
A_5	(Any)	$\{A_{30}\}$
A_6	(Any)	$\{A_{30}\}$
A_7	(Any)	$\{A_{30}\}$
A_8	(Any)	$\{A_{30}\}$
A_9	(Any)	$\{A_{30}\}$
A_{10}	(Any)	$\{A_{30}\}$

$S' \in \overline{S}$. Observe that $\overline{S} = \emptyset$ when no dependency is established for S and U.

After receiving VC_U^S, U stores it in their digital wallet so that it can be provided to another system during the dependencies verification phase.

Table 4
Voided authentication attributes (2/2).

Attribute A	Value v	*voided-by*(A, v)
A_{11}	cleartext	$\{A_{12}, A_{13}, A_{30}\}$
	one-way function	$\{A_{12}, A_{13}, A_{30}\}$
	salted one-way function	$\{A_{13}, A_{30}\}$
	one-way key derivation function	$\{A_{30}\}$
A_{12}	(Any)	$\{A_{30}\}$
A_{13}	(Any)	$\{A_{30}\}$
A_{14}	Security questions	$\{A_{15}, \dots, A_{18}, A_{30}\}$
	URL token	$\{A_{18}, A_{30}\}$
	PINs	$\{A_{15}, A_{16}, A_{17}, A_{30}\}$
	Offline methods	$\{A_{15}, \dots, A_{18}, A_{30}\}$
A_{15}	(Any)	$\{A_{18}, A_{30}\}$
A_{16}	(Any)	$\{A_{18}, A_{30}\}$
A_{17}	(Any)	$\{A_{18}, A_{30}\}$
A_{18}	(Any)	$\{A_{15}, A_{16}, A_{17}, A_{30}\}$
A_{19}	(Any)	$\{A_{24}, \dots, A_{32}\}$
A_{20}	(Any)	$\{A_{24}, \dots, A_{32}\}$
A_{21}	cleartext	$\{A_{22}, \dots, A_{32}\}$
	one-way function	$\{A_{22}, \dots, A_{32}\}$
	salted one-way function	$\{A_{23}, \dots, A_{30}\}$
	one-way key derivation function	$\{A_{24}, \dots, A_{32}\}$
A_{22}	(Any)	$\{A_{24}, \dots, A_{32}\}$
A_{23}	(Any)	$\{A_{24}, \dots, A_{32}\}$
A_{24}	(Any)	$\{A_{27}, \dots, A_{32}\}$
A_{25}	(Any)	$\{A_{27}, \dots, A_{32}\}$
A_{26}	(Any)	$\{A_{27}, \dots, A_{32}\}$
A_{27}	(Any)	$\{A_{19}, \dots, A_{26}\}$
A_{28}	(Any)	$\{A_{19}, \dots, A_{26}\}$
A_{29}	(Any)	$\{A_{19}, \dots, A_{26}\}$
A_{30}	(Any)	$\{A_{19}, \dots, A_{26}\}$
A_{31}	(Any)	$\{A_{19}, \dots, A_{26}, A_{32}\}$
A_{32}	(Any)	$\{A_{19}, \dots, A_{26}, A_{31}\}$

User registration and dependency verification The user registration phase is performed as described in Section 3 and does not require the exchange of verifiable credentials.

On the other hand, an SSI-based protocol that implements Algorithm VERIFY-DEPENDENCY includes the following steps.

1. S contacts the endpoint of S' to obtain $VC_{S'}$.
2. After performing the SSI-verification of $VC_{S'}$ (see the Verification Process in Section 6.2), S retrieves (from the credential itself) *sys-val*(S', A) for each $A \in \mathcal{A}_{sys}$.
3. U provides S with $VC_U^{S'}$, stored in their digital wallet.
4. After performing the SSI-verification of $VC_U^{S'}$, S retrieves (from the credential itself) *usr-val*(S', A, U) for each $A \in \mathcal{A}_{usr}$.

5. For each $A \in \mathcal{A}_{auth}$, S checks that A is compliant with $policy(S, A)$. This is performed by Algorithm VERIFY-FORWARD (Lines 1–9).
6. S retrieves the set $\overline{S} = \{S'' \in \mathcal{S} : S' \overset{U}{\longrightarrow} S''\}$ from $VC_U^{S'}$. Then for each $S'' \in \overline{S}$, S repeats Steps 1–6 with S'' in place of S' (if S'' has been already contacted by S, the procedure ends).
7. S retrieves the set $\hat{S} = \{S'' \in \mathcal{S} : S'' \overset{U}{\longrightarrow} S\}$ (stored by S when each $S'' \in \hat{S}$ contacts S to establish a dependency). Then, for each $S'' \in \hat{S}$, S contacts S'' by providing the endpoint of S'. Finally, S'' (in place of S) repeats Steps 1–7 with S'.

From a technological point of view, in Steps 1 and 3, the credentials can be exchanged off-chain through the DIDComm protocol (Hardman, 2019) supported by Trinsic. In addition, in Step 4, selective disclosure and ZKP proof mechanisms can be adopted. Specifically, for each attribute $A \in \mathcal{A}_{usr}$ such that $policy(S, A) = \{(min\text{-}val(A), \emptyset)\}$ (i.e., no constraint for A), S does not require the claim related to A. Then, the selective disclosure mechanism allows U to provide S with just the values for the attributes needed to satisfy the policy of S.

Another issue concerns the presence of sensitive user attributes whose exact value should not be revealed. For example, "Password length" is a sensitive attribute since the actual length of the password should not be revealed. On the other hand, for a system, it is typically sufficient to know that the password length is in a specific range (e.g., greater than a given number of characters). In this case, ZKP proofs can be forged by U starting from the credentials of U to prove that the value of an attribute is in a range without providing the exact value.

Login procedure We assume that during the user registration and dependency verification process, S locally stores all the VCs provided by both the user U and all the systems S depends on. Then, every time U authenticates with S, the latter should verify if any of the credentials has been revoked. If this is the case, then S performs the process described in the previous section again, in order to retrieve the new credentials and check whether they are compliant with its policy.

Attribute revocation Suppose the value of a user attribute A, for a system S, changes over time. Two cases can occur:

- The new $usr\text{-}val(S, A, U)$ is greater than the old one. In this case, S should provide U with a new credential VC_U^S with an updated claim A. The old credential is not revoked so it can still be verified during the login phase. On the other hand, the new credential can be used by U with other systems, to establish new dependencies.
- The new $usr\text{-}val(S, A, U)$ is smaller than the old one. In this case, S should revoke the old VC_U^S and issue a new credential. When the user attempts to login with a system that depends on S, since the credential is not valid, the user registration and dependency verification is repeated as described for the login procedure.

6.4. *Identification attributes*

The discussion carried out so far is focused on the authentication domain — we assume that a system depends on another when the former relies on the latter for an authentication procedure, such as in the cases of federated authentication and authenticator recovery.

It could also be important to consider dependencies that are established when a system relies on another for the *identification* process (possibly, together with authentication). For instance, the *Facebook Login Plugin* allows users to authenticate to other systems using their Facebook credentials. When a user authenticates, Facebook provides additional attributes (e.g., full name, hometown, and birthday), which are in turn used to identify the user. In these cases, a system might want to define a policy that includes its requirements regarding user identification.

Table 5
Reference identification attributes.

Attribute A	Name	$values(A)$
A_1	"Presence verification"	⟨No, remote unsupervised, remote supervised, in presence⟩
A_2	"Address confirmation proofing"	⟨no, remote, in person⟩
A_3	"In person enrollment code validity (days)"	⟨7,. . . ,1⟩
A_4	"Means to send remote enrollment code"	⟨email, telephone, postal address⟩
A_5	"Enrollment code reset after first use"	⟨false, true⟩
A_6	"Biometric collection"	⟨false, true⟩
A_7	"Number of consecutive failed attemps in biometric systems"	⟨2,. . . ,10⟩
A_8	"Presentation Attack Detection (PAD) implementation in biometric systems"	⟨false, true⟩
A_9	"Throttling applied to biometric systems"	⟨false, true⟩

To this aim, our proposed framework can be easily extended to the identification domain. Specifically, we can define a set $\mathcal{A}_{id}$ of identification attributes and re-apply the definitions of policies (Section 2.2) and dependencies (Section 4.1) after just replacing $\mathcal{A}_{auth}$ with $\mathcal{A}_{id}$. Hence, for each $A \in \mathcal{A}_{auth}$, each system S additionally defines $policy_{id}(A, S)$ for each $A \in \mathcal{A}_{id}$. The verification process is then applied when a dependency $S \overset{U}{\longrightarrow}_{id} S'$ in the identification domain has to be established.

Table 5 reports a possible reference set of identification attributes derived from the *NIST Digital Identity Guidelines: Enrollment and Identity Proofing Requirements* (NIST, 2017b).

7. Related work

Interactions among authentication systems often happen when the authenticator used for a system is lost (i.e., fallback authentication (Al-Husain and Alkhalifah, 2021)), when a "main" system delegates authentication to a "secondary" system (e.g., federated authentication Boehm et al., 2008), or when a user relies on the same secret (e.g., password, PIN, and so on) on different systems (Das et al., 2014; Ives et al., 2004).

From a security point of view, such cases (and many other ones) are equivalent, since a vulnerability in the secondary system may be exploited to compromise the main one. This problem is well acknowledged in the literature (NIST, 2017a; Hang, 2016) — however, to the best of our knowledge, no fully suitable solution is currently available.

A lot of related work focuses on security issues in authentication systems (Ross et al., 2005; Zhao and Li, 2007; Shirvanian et al., 2021; Buccafurri et al., 2022; Binbeshr et al., 2021; Zimmermann and Gerber, 2020; Javed et al., 2014; Alomar et al., 2017). However, they are mostly focused on specific authenticator types and do not consider possible interactions among different systems.

Zimmermann and Gerber (2020); Quermann et al. (2018); Markert et al. (2020); Stavova et al. (2016) focus on examining existing fallback authentication systems in the wild. For instance, Quermann et al. (2018) reports authentication methods adopted in 48 different services, including websites, IoT/smart home devices, and mobile devices. The paper shows that one of the most common fallback authentication systems for recovering users' passwords still consists in answering security questions. The latter are known to be unsuitable as recovery mechanisms — they have important security issues (Rabkin, 2008; Sagar and Waghmare, 2016) that make them exploitable to bypass the main authentication system. Markert et al. (2020) makes a long-term comparative analysis of the usability provided by five fallback authentication systems (email, SMS, security questions, designated trustees, and browser fingerprint). No comparison from the security point of view is provided. Maqbali and Mitchell (2019) provides an overview of the security issues for all the mechanisms commonly adopted to recover users' passwords. Overall, these works only consider a relatively narrow set of methods for fallback authentication.

Other works are aimed at proposing frameworks to compare authentication schemes. They are related to ours since the definition of an order among authentication systems, capturing security capabilities, may prevent a system from relying on a less secure one. Among these proposals, the most relevant ones are (Bonneau et al., 2012; Velásquez et al., 2018). Specifically, Bonneau et al. (2012) is considered the state-of-the-art framework for comparing authentication systems. The authors consider 25 requirements, along three main categories (usability, deployability, and security). They evaluate 35 authentication mechanisms and, for each mechanism, classify each requirement as satisfied, partially satisfied, or not satisfied. However, as the authors themselves acknowledge, an important limitation of the proposed approach is that all the requirements are treated equally when comparing mechanisms, while instead, to be more realistic, different requirements should have different weights — without the latter, authentication mechanisms appear still hard to compare in real settings. Velásquez et al. (2018), along these lines, is aimed at finding the “best” authentication system for a given application. However, the requirements defined by the authors are relatively vague, and the approach based on assigning weights to each of them is not clearly defined.

None of the above works *directly* address the security issues related to *interacting* authentication systems. Moreover, all the above frameworks present important practical limitations in the comparison strategies. Overcoming the above limitations, also by enabling transitivity of dependencies and by introducing a way to compare system policies (and thus authentication systems), as we have discussed in Section 4.3, is an important direction for future work.

Differently from the domain of authentication systems, the problem of enforcing security policies when dealing with interacting systems is widely covered in the domain of web services composition (Boumlik and Mejri, 2019; Ranchal et al., 2018). However, these solutions cannot find direct applications in our domain since they are based on a centralized policy definition. On the contrary, we deal with a more challenging situation in which each interacting system specifies its own policy. Thus, in this paper we proposed a solution tailored to this context.

8. Conclusions and future work

This paper proposes a formal approach to the problem of interacting authentication systems and provides an SSI-based implementation. Future work will be devoted to issues related to real-life adoption of a system based on our solution – specifically, those discussed in Section 6.3 regarding the way in which verifiable credentials of system authentication attributes are released. Indeed, a more general perspective could be considered, framing the entire solution in a business/market setting, possibly combining it with a trust and reputation (decentralized) based approach. In particular, a detailed business model, based on incentives, trust, and reputation can be designed, to provide new insights and possibly highlight a fruitful integration of our approach with the European framework of digital identities outlined by the current evolution of the eIDAS regulation (eIDAS, 2014).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU.

References

AlHusain, R., Alkhalifah, A., 2021. Evaluating fallback authentication research: a systematic literature review. Comput. Secur. 111, 102487.
Alomar, N., Alsaleh, M., Alarifi, A., 2017. Social authentication applications, attacks, defense strategies and future research directions: a systematic review. IEEE Commun. Surv. Tutor. 19 (2), 1080–1111.
Alshamrani, A., Myneni, S., Chowdhary, A., Huang, D., 2019. A survey on advanced persistent threats: techniques, solutions, challenges, and research opportunities. IEEE Commun. Surv. Tutor. 21 (2), 1851–1877. https://doi.org/10.1109/COMST.2019.2891891.
Binbeshr, F., Kiah, M.L.M., Por, L.Y., Zaidan, A.A., 2021. A systematic review of pin-entry methods resistant to shoulder-surfing attacks. Comput. Secur. 101, 102116.
Boehm, O., Caumanns, J., Franke, M., Pfaff, O., 2008. Federated authentication and authorization: a case study. In: 12th International IEEE Enterprise Distributed Object Computing Conference. ECOC 2008, 15-19 September 2008, Munich, Germany. IEEE Computer Society, pp. 356–362.
Bonneau, J., Herley, C., van Oorschot, P.C., Stajano, F., 2012. The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. In: IEEE Symposium on Security and Privacy. SP 2012, 21-23 May 2012, San Francisco, California, USA. IEEE Computer Society, pp. 553–567.
Boumlik, L., Mejri, M., 2019. Security enforcement on web services compositions. In: 2019 IEEE Symposium on Computers and Communications (ISCC). IEEE, pp. 1010–1015.
Buccafurri, F., De Angelis, V., Lazzaro, S., 2022. The ginger: another spice to hinder attacks on password files. In: Proceedings of the 18th International Conference on Web Information Systems and Technologies. WEBIST 2022, Valletta, Malta, October 25-27, 2022. SCITEPRESS, pp. 166–173.
Campobasso, M., Allodi, L., 2020. Impersonation-as-a-service: characterizing the emerging criminal infrastructure for user impersonation at scale. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1665–1680.
Das, A., Bonneau, J., Caesar, M., Borisov, N., Wang, X., 2014. The tangled web of password reuse. In: 21st Annual Network and Distributed System Security Symposium. NDSS 2014, San Diego, California, USA, February 23-26, 2014. The Internet Society.
Davie, M., Gisolfi, D., Hardman, D., Jordan, J., O'Donnell, D., Reed, D., 2019. The trust over IP stack. IEEE Commun. Stand. Mag. 3 (4), 46–51.
eIDAS, 2014. eIDAS: electronic IDentification, Authentication and trust Services. http://data.europa.eu/eli/reg/2014/910/oj. (Accessed 20 February 2024).
Hang, A., 2016. Exploiting autobiographical memory for fallback authentication on smartphones. Ph.D. thesis. LMU Munchen.
Hardman, D., 2019. Aries RFC 0005: DID communication. https://github.com/hyperledger/aries-rfcs/blob/main/concepts/0005-didcomm/README.md. (Accessed 20 February 2024).
Ives, B., Walsh, K.R., Schneider, H., 2004. The domino effect of password reuse. Commun. ACM 47 (4), 75–78.
Javed, A., Bletgen, D., Kohlar, F., Dürmuth, M., Schwenk, J., 2014. Secure fallback authentication and the trusted friend attack. In: 34th International Conference on Distributed Computing Systems Workshops (ICDCS 2014 Workshops). Madrid, Spain, June 30 - July 3, 2014. IEEE Computer Society, pp. 22–28.
Li, Y., Dai, W., Bai, J., Gan, X., Wang, J., Wang, X., 2018. An intelligence-driven security-aware defense mechanism for advanced persistent threats. IEEE Trans. Inf. Forensics Secur. 14 (3), 646–661.
Maqbali, F.A., Mitchell, C.J., 2019. Web password recovery: a necessary evil? In: Proceedings of the Future Technologies Conference (FTC). Springer, pp. 324–341.
Markert, P., Golla, M., Stobert, E., Dürmuth, M., 2020. A comparative long-term study of fallback authentication. In: 27th Annual Network and Distributed System Security Symposium. The Internet Society.
Mühle, A., Grüner, A., Gayvoronskaya, T., Meinel, C., 2018. A survey on essential components of a self-sovereign identity. Comput. Sci. Rev. 30, 80–86.
Mukta, R., Martens, J., Paik, H., Lu, Q., Kanhere, S.S., 2020. Blockchain-based verifiable credential sharing with selective disclosure. In: 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2020. TrustCom 2020, Guangzhou, China, December 29, 2020 - January 1, 2021. IEEE, pp. 959–966.
NIST, 2017a. NIST special publication 800-63b: Digital identity guidelines – authentication and lifecycle management. https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-63b.pdf. (Accessed 20 February 2024).

NIST, 2017b. NIST special publication 800-63a: Digital identity guidelines – enrollment and identity proofing. https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-63a.pdf. (Accessed 20 February 2024).

Oberle, A., Larbig, P., Marx, R., Weber, F.G., Scheuermann, D., Fages, D., Thomas, F., 2016. Preventing pass-the-hash and similar impersonation attacks in enterprise infrastructures. In: 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), pp. 800–807.

OWASP, 2024. OWASP authentication cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html. (Accessed 20 February 2024).

Preukschat, A., Reed, D., 2021. Self-Sovereign Identity. Manning Publications.

Quermann, N., Harbach, M., Dürmuth, M., 2018. The state of user authentication in the wild. In: Who Are You? Adventures in Authentication Workshop 2018.

Rabkin, A., 2008. Personal knowledge questions for fallback authentication: security questions in the era of Facebook. In: Proceedings of the 4th Symposium on Usable Privacy and Security. SOUPS 2008, Pittsburgh, Pennsylvania, USA, July 23-25, 2008. In: ACM International Conference Proceeding Series. ACM, pp. 13–23.

Ranchal, R., Bhargava, B., Angin, P., ben Othmane, L., 2018. Epics: a framework for enforcing security policies in composite web services. IEEE Trans. Serv. Comput. 12 (3), 415–428.

Ross, B., Jackson, C., Miyake, N., Boneh, D., Mitchell, J.C., 2005. Stronger password authentication using browser extensions. In: Proceedings of the 14th USENIX Security Symposium. Baltimore, MD, USA, July 31 - August 5, 2005.

Sagar, K., Waghmare, V., 2016. Measuring the security and reliability of authentication of social networking sites. Proc. Comput. Sci. 79, 668–674.

Sakimura, N., Bradley, J., Jones, M., De Medeiros, B., Mortimore, C., 2014. Openid connect core 1.0. The OpenID Foundation, p. S3.

Shirvanian, M., Price, C.R., Jubur, M., Saxena, N., Jarecki, S., Krawczyk, H., 2021. A hidden-password online password manager. In: SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event. Republic of Korea, March 22–26, 2021. ACM, pp. 1683–1686.

Squicciarini, A.C., Bhargav-Spantzel, A., Bertino, E., Czeksis, A.B., 2007. Auth-sl-a system for the specification and enforcement of quality-based authentication policies. In: Information and Communications Security: 9th International Conference, ICICS 2007, Zhengzhou, China, December 12-15, 2007. Proceedings 9. Springer, pp. 386–397.

Stavova, V., Matyas, V., Just, M., 2016. Codes v. people: a comparative usability study of two password recovery mechanisms. In: Information Security Theory and Practice - 10th IFIP WG 11.2 International Conference. WISTP 2016, Heraklion, Crete, Greece, September 26-27, 2016, Proceedings. In: Lecture Notes in Computer Science, vol. 9895. Springer, pp. 35–50.

Tobin, A., Reed, D., 2016. The inevitable rise of self-sovereign identity. Sovrin Found. 29 (2016), 18.

Trinsic, 2024. https://github.com/trinsic-id. (Accessed 20 February 2024).

Velásquez, I., Caro, A., Rodríguez Kontun, A., 2018. A framework for recommendation of authentication schemes and methods. Inf. Softw. Technol. 96, 27–37.

Wilson, Y., Hingnikar, A., 2019. Solving Identity Management in Modern Applications: Demystifying OAuth 2.0, OpenID Connect, and SAML 2.0. Springer.

W3C, 2022a. Decentralized Identifiers (DIDs) v1.0. https://www.w3.org/TR/did-core. (Accessed 20 February 2024).

W3C, 2022b. Verifiable Credentials Data Model v1.1. https://www.w3.org/TR/vc-data-model. (Accessed 20 February 2024).

Zhao, H., Li, X., 2007. S3PAS: a scalable shoulder-surfing resistant textual-graphical password authentication scheme. In: 21st International Conference on Advanced Information Networking and Applications. AINA 2007, Workshops Proceedings, Volume 2, May 21-23, 2007, Niagara Falls, Canada. IEEE Computer Society, pp. 467–472.

Zimmermann, V., Gerber, N., 2020. The password is dead, long live the password - a laboratory study on user perceptions of authentication schemes. Int. J. Hum.-Comput. Stud. 133, 26–44.