Flexible and Survivable Single Sign-On

Federico Magnanini[1], Luca Ferretti[1], and Michele Colajanni[2]

[1] University of Modena and Reggio Emilia, Italy,
{luca.ferretti,federico.magnanini}@unimore.it
[2] University of Bologna, Italy, michele.colajanni@unibo.it

Abstract. Single sign-on (SSO) is a popular authentication method that is vulnerable to attacks exploiting the single points of failure of its centralized design. This problem is addressed by survivable SSO protocols relying on distributed architectures that enable a set of servers to collectively authenticate a user. However, existing survivable SSO protocols have limitations because they do not allow service providers to modify security parameters after protocol setup. This paper introduces the first survivable SSO protocol that guarantees flexibility. This property is of utmost importance for SSO because it allows service providers to tailor the trade-off between performance overhead and security requirements of multiple services and even to preserve compatibility with non-survivable SSO.

Keywords: Survivability, Single Sign-On, Distributed systems, Cloud, Flexibility

1 Introduction

Single sign-on (SSO) is a popular protocol to authenticate users requiring access to multiple Web-based services. Typically, an identity provider manages one logical identity server that issues authentication tokens proving user identity to service providers. The centralized design of SSO protocols is vulnerable to authentication tokens forgery as demonstrated by recent incidents [10,2] where cyber attackers compromised the identity server and forged tokens that falsely impersonated users towards service providers. This issue can be addressed by so-called *survivable SSO protocols* that can prevent user impersonation even in presence of attacks (e.g., [3,6,26]). In survivable SSO, the identity provider manages multiple identity servers. A user authenticates himself to a subset of identity servers that collectively sign an authentication token and demonstrate the user identity to service providers. The amount of signing identity servers must be greater than a security threshold which defines the maximum number of identity servers that the adversary can violate.

Existing proposals achieve survivable token release by signing tokens through threshold signatures, and survivable user authentication through distributed password-based authentication protocols [3,6,26]. The problem is that threshold signatures tend to be unrealistic in practice because they do not guarantee

flexibility. They prevent service providers from dynamically adjusting the value of the security threshold during protocol execution and they are not backwards-compatible with existing SSO systems. The lack of flexibility and of backwards compatibility prevent service providers from offering services with different identity assurance levels [1,16], and from the possibility of dynamically adjusting the threshold based on user contextual information as suggested by the recent zero trust paradigm [25].

We propose an original survivable SSO protocol where survivable token release is achieved by signing authentication tokens through conventional digital signatures instead of threshold signatures as in literature. This approach enables the design of a survivable token release scheme that guarantees flexibility and preserves backwards compatibility with non-survivable SSO solutions. Moreover, we show that it is possible to guarantee survivable user authentication through password-based protocols even if they are not designed for distributed architectures. We evaluate the security of the proposed token release scheme and show the security of the overall SSO protocol by considering existing password-based authentication methods.

The practical relevance of the proposal is twofold. First, flexible and survivable SSO can be leveraged to mitigate emerging attacks to access confidential cloud resources through rogue authentication tokens [10]. Moreover, the high security level that is guaranteed by flexible and survivable SSO protocols is a perfect combination for mission critical systems requiring robust and reliable authentication mechanisms even under attack [13].

The paper is organized as follows. Section 2 discusses related work. Section 3 describes the system model and the single sign-on framework. Section 4 describes the threat model. Section 5 discusses the details and guarantees of the proposed novel survivable token release scheme. Section 6 shows the security level of the proposal. Section 7 evaluates the security and flexibility of the proposal and of related SSO protocols when integrated with different credential management systems. Section 8 highlights final remarks.

2 Related Work

This paper is related to recent results investigating SSO protocols with different trade-offs between survivable security guarantees and flexible configuration [3,6,26]. For example, the authors in [3] propose a SSO protocol that tolerates the violation of up to a threshold of identity servers. The value of the threshold can be configured at setup time to tolerate from one compromised identity server up to a dishonest majority including a single honest identity server. The problem is that this protocol does not guarantee recoverability because it does not include the due procedures to recover compromised identity servers to a safe state. As a result, the protocol cannot be considered survivable because recoverability is a mandatory security guarantee in these systems [5]. We propose a protocol that guarantees survivability by defining specific procedures to recover compromised identity servers.

The guarantee of survivability is also analyzed by the proposal of a password-based survivable SSO protocol [6]. The authors consider a strong adversarial model that requires to trade token unforgeability for availability, as the protocol does not terminate if a single identity server is unavailable. We assume a different model called *mobile adversarial model* [17]. Although it is weaker than that considered in [6], the mobile adversary is a realistic model for SSO scenarios and allows the proposed protocol to guarantee termination even in presence of a fully malicious minority of identity servers. The authors of [6] achieve survivable token release by signing authentication tokens through an original RSA-based threshold signature scheme. However, the lack of flexibility of threshold signatures represents a major problem as they force the identity provider to set the value of the security threshold at setup time. Moreover, threshold signatures cause management issues as they cannot guarantee backwards compatibility with non-survivable SSO protocols.

The proposed protocol guarantees flexible SSO and offers the possibility of choosing the value of the security threshold at verification time. This allows a service provider to offer multiple services with different identity assurance levels (e.g., [1,16]) and to choose the most suitable threshold for each of them. Moreover, it is possible to dynamically adjust the security threshold depending on user contextual information, as suggested by the zero trust paradigm [25], and to tailor the best trade-off between performance and security for each service. Finally, the proposed flexible and survivable SSO can preserve compatibility with non-survivable SSO. This would enable a gradual transition of existing service providers towards survivable SSO. A service provider can enable backwards compatible support to survivable SSO by updating the authentication token verification algorithm.

The password-based survivable SSO proposed in [26] obtains a better trade-off in terms of threshold configuration and survivability with respect to [3] and [6]. It allows the configuration of the security threshold at setup time and guarantees survivability. Although the authors do not explicitly specify an adversarial model, their proposal seems to consider a mobile adversary similar to that proposed in this paper. While their protocol obtains good trade-offs, it lacks flexibility due to the adoption of threshold signatures during the token authentication phase.

3 System Model

We describe the survivable SSO protocol by referring to Figure 1 showing the main entities, data and operation flows. The proposed protocol involves four entities: *user*, *service provider*, *identity provider*, and a set of *identity servers*. The user denotes a person who wants to access services and resources maintained by the service provider. The identity provider denotes an authority that defines and operates a set of identity servers to offer the survivable single sign-on protocol. The protocol involves the following types of data:

- *user credentials*: unique information held by each user presented to identity servers for authentication;
- *credentials databases*: data structures independently maintained by identity servers to verify users credentials;
- *partial tokens*: assertions about users identities authenticated by a single identity server;
- *authentication tokens*: assertions about users identities whose authenticity is guaranteed by a subset of identity servers;
- *token signing keys*: secret cryptographic material held by each identity server to authenticate authentication tokens;
- *identity provider certificate*: public cryptographic material used by the service provider to verify authentication tokens;
- *identity provider signing key*: secret cryptographic material used by the identity provider to authenticate the identity provider certificate.

The proposed protocol consists of five operations.

- *Setup*: the identity provider defines the initial set of identity servers, and releases a certificate that authenticates identity servers public keys. We assume that the certificate is distributed to all actors by using orthogonal public key distribution protocols [8];
- *Register*: the user registers his credentials to all identity servers credentials databases;
- *Sign-on*: the user requests an authentication token to identity servers. This operation is composed by the following steps:
  - *Verify credential*: an identity server verifies user credentials against his credential database;

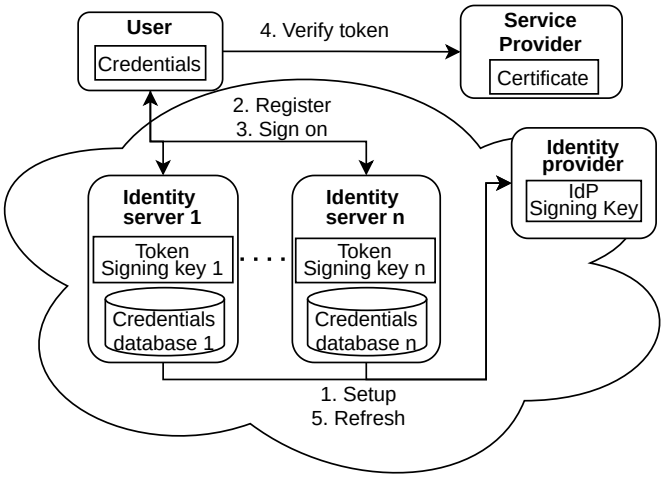

Fig. 1. Architecture and high level protocol flow

  - *Release partial*: an identity server releases a partial token to an authenticated user;
  - *Combine*: the user combines the partial tokens collected by a threshold of identity servers in an authentication token;
- *Verify token*: the service provider uses the identity provider certificate to verify the authenticity and validity of the authentication token presented by the user;
- *Refresh*: the identity provider updates identity servers secret cryptographic material.

The protocol requires that users establish secure (confidential and authenticated) bidirectional communication channels with legitimate identity servers by using public keys of identity servers distributed during the *Setup* phase. Communication channels allow users to authenticate identity servers, and not vice versa, as it is common for standard HTTPS communications.

The proposed framework represents a novel contribution to model the operations of survivable single sign-on protocols. It captures common operations shared by related proposals which were not highlighted by previous literature, and allows us to compare the proposed protocol with related works [3,6,26]. The framework extends existing non-survivable single sign-on frameworks by introducing the Release partial and Combine procedures. Existing related survivable SSO protocols adopt a similar system model where identity servers are coordinated through decentralized protocols which do not require an identity provider [3,6,26]. However, they do not guarantee flexibility (see Section 6). The proposed architecture introduces the additional role of the identity provider because the considered scenarios, such as cloud-based SSO, are characterized by centralized governance where the identity provider acts as an authority that operates identity servers during Setup and Refresh operations. At Setup time, the identity provider defines the infrastructure while at Refresh time it proactively secures it.

The proposal does not limit the identity provider from being a decentralized entity because it can be extended to support decentralized execution of the Setup and Refresh operations by leveraging ideas from [21]. For ease of presentation and without loss of generality, in the remainder of this paper we consider the identity provider as one entity.

We enable identity providers to offer flexible and survivable SSO as a service. An identity provider can execute the Setup operation by defining the maximum security threshold k_{max} and by provisioning an infrastructure of $2k_{max}+1$ failure-independent identity servers. Service providers can choose the most appropriate security threshold value between zero and k_{max} to enforce survivability on their services. A security threshold equal to zero maintains compatibility with existing non-survivable SSO. A security threshold equal to k_{max} allows service providers to enforce the highest level of identity assurance even in presence of k_{max} compromised identity servers. Guaranteeing a practical failure independence of all servers against benign and malicious faults tends to become quickly an intractable challenge as demonstrated in [15,19]. Hence, we can assume that

in practice the value of k_{max} is at most of few units. If we accept stronger security assumptions on identity servers, then the value of k_{max} can be increased above the few units. For example, some identity servers may share the same operating system. This security trade-off simplifies the technological challenge of provisioning and maintaining several operating systems to enable the deployment of a larger yet less diverse set of identity servers.

4 Threat model

We discuss possible attacks from a twofold perspective: we discuss violation and recovery patterns throughout the protocol lifetime; we analyze the multiple classes of attacks that an adversary can adopt to subvert the protocol security. We use these analyses to assess the security guarantees of the proposal in Sections 6 and 7.

For the threat analysis, we consider the popular *mobile adversary model* that aims at subverting the survivable single sign-on protocol. This model was proposed in the context of distributed function evaluation [23] and applied to secret sharing [17], multiparty computation [12], intrusion tolerant certification authorities [27], key management systems [11], cloud-based secure logging [24] and secure software update systems [20]. It assumes that identity servers are failure-independent and that at any instant an identity server is either *honest* or *compromised*. A compromised identity server can be recovered and become honest after that its hardware and software have been reset to a known clean state and its secret cryptographic material has been obsoleted. A recovery of all identity servers is periodically executed by the identity provider during the Refresh operation. The proposed protocol tolerates that a minority of identity servers is compromised simultaneously, that is, it guarantees security against adversaries that violate up to $k < n/2$ identity servers, where n is the total number of identity servers. This is the typical security level guaranteed by related works using the mobile adversary model (e.g., [17,11]).

We discuss violation and recovery patterns between periodic Refresh operations by referring to Figure 2. The time horizon is divided in *time periods*, where each begins with the execution of the Refresh operation. We denote the remaining part of the time period after the completion of the Refresh as *operation period*. During the operation period, honest identity servers operate the single sign-on protocol according to initial specification. Each highlighted area represents a time interval during which the adversary has compromised up to k identity servers.

The mobile adversary model considers four attack patterns. The first pattern captures an adversary that has corrupted up to k identity servers during the operation period and that is removed by the identity provider during the next Refresh operation. The second pattern considers an adversary that has corrupted up to k identity servers during the Refresh operation. The adversary has the additional power to interfere with the identity provider during Refresh. The third and fourth patterns capture a powerful and elusive adversary that is able

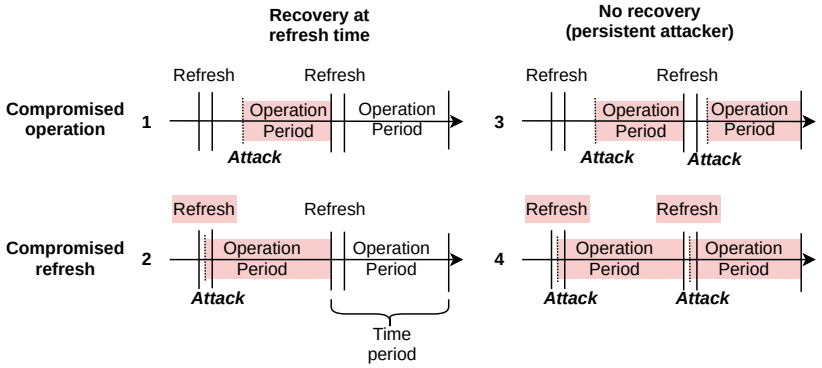

Fig. 2. Violation patterns

to move laterally among identity servers and ensures a persistent presence at the identity provider infrastructure even after Refresh operations. An attacker cannot control more than k identity servers during each period.

An adversary can perform different types of attacks to subvert the survivable single sign-on protocol. We label attacks to reference each of them when analyzing security guarantees in Sections 6 and 7.

- **A.** Violation of the token release system within identity servers:
  - **A1**: the adversary violates the confidentiality of the token signing keys within identity servers to forge authentication tokens;
  - **A2**: the adversary violates the integrity of the token release system by returning bogus partial tokens to users;
  - **A3**: the adversary violates the availability of the token release system by deleting token signing keys or by not returning partial tokens to users.
- **B.** Violation of the identity verification protocols within identity servers:
  - **B1**: the adversary compromises the integrity of the credentials database of an identity server to forcefully set known credentials to users accounts;
  - **B2**: the adversary violates the availability of the identity verification protocol by deleting the credential database or by not completing credential verification;
  - **B3**: the adversary violates the confidentiality of the credentials database of an identity server to recover users credentials;
  - **B4**: the adversary reads an identity server internal state during credential verification to recover users credentials;
  - **B5**: the adversary executes Man-in-The-Middle attacks from compromised identity servers to impersonate the user at honest identity servers during the Sign-on operation.
- **C.** Violation of the identity provider:
  - **C1**: the adversary violates the confidentiality of the identity provider signing key.

5 Survivable token release

A comprehensive analysis of the proposed SSO protocol is in Section 7. Here, we focus on the token release scheme. The key insight of the proposed token release scheme is twofold. First, signing authentication tokens through conventional digital signatures allows to achieve flexibility. Second, proactively rotating token signing keys allows to guarantee security against lateral movement of the mobile adversary while the identity provider allows to efficiently authenticate rotated public keys and modified system parameters. We consider a black box digital signature scheme defined by the following operations framework:

- $\langle sk, pk \rangle \leftarrow \mathsf{KeyGen}()$: generate secret key sk and public key pk;
- $\sigma \leftarrow \mathsf{Sign}(sk, m)$: compute signature σ on message m with secret key sk;
- $0 \vee 1 \leftarrow \mathsf{Verify}(pk, m, \sigma)$: if signature σ authenticates message m under public key pk output 1, 0 otherwise.

We assume that the identity provider has established the security level of the adopted cryptographic schemes and that the resulting public parameters are known to all actors. For ease of notation, we omit public parameters from the scheme operations. The proposed token release scheme implements the following operations.

$crt \leftarrow \mathsf{Setup}(sk_{IdP}, k_{max}, \{\langle pk_1, \pi_1 \rangle \ldots, \langle pk_n, \pi_n \rangle\})$. The identity provider, given his secret key sk_{IdP} and security threshold k_{max} verifies the identity servers certificate signing requests $\{\pi_1, \ldots, \pi_n\}$ and authenticates the corresponding public keys $\{pk_1, \ldots, pk_n\}$ by computing the identity provider certificate crt. To this aim it executes the following steps:

- the service provider defines the value k_{max} and a set of $n = 2k_{max} + 1$ identity servers;
- each identity server executes the $\mathsf{KeyGen}()$ algorithm to compute the signing key pair $\langle sk_i, pk_i \rangle$;
- each identity server i then computes a certificate signing request π_i for pk_i, and sends π_i to the identity provider over a secure channel. Certificate signing requests can be computed with well-established standard algorithms [22].
- the identity provider collects the public keys $PK = \{pk_1, \ldots, pk_n\}$ and verifies each certificate signing request in $\{\pi_i\}$;
- if all received certificate signing requests $\{\pi_i\}$ are valid, the identity provider authenticates the corresponding set of public keys and the value k_{max} by signing the pair $\langle PK, k_{max} \rangle$ with his private key sk_{IdP}, producing $crt = \langle PK, k_{max}, \sigma_{crt} \rangle$;
- the identity provider publicly distributes crt.

The authentication methods used in procedures Register and VerifyCredential are orthogonal to the proposed token release scheme in terms of functionality, hence we can omit details. A comprehensive discussion about integration with different user authentication methods is in Section 7.

$\sigma_i \leftarrow \mathsf{ReleasePartial}(sk_i, id)$: identity server i with secret key sk_i, given identity id produces partial token σ_i. The ReleasePartial procedure assumes that

identity id has been already verified during VerifyCredential. The identity server signs id with his secret key sk_i by executing algorithm $\mathsf{Sign}(sk_i, id)$ of the signature scheme. The identity server then sends the resulting signature σ_i to the user over a secure channel.

$\langle id, \{\sigma_i\}, L\rangle \leftarrow \mathsf{Combine}(id, \{\sigma_i\}_{i\in L})$: the user verifies partial tokens $\{\sigma_i\}_{i\in L}$ collected from the set of identity servers L, and outputs the authentication token $\langle id, \{\sigma_i\}, L\rangle$. To this aim he executes the following steps:

- when the user has collected $k+1$ partial tokens, as required by the service provider, the user determines the set of servers L that produced the collected partial tokens;
- the user then verifies that each of the collected partial tokens is authentic, by executing the $\mathsf{Verify}(pk_i, id, \sigma_i)$ for each $i \in L$;
- if all partial tokens are authentic, the user outputs the authentication token $\langle id, \{\sigma_i\}, L\rangle$.

$0 \vee 1 \leftarrow \mathsf{VerifyToken}(\langle id, \{\sigma_i\}, L\rangle, crt, k)$: the service provider verifies whether all signatures in $\{\sigma_i\}$, produced by the set of identity servers L, authenticate id by using crt. It outputs 0 if any σ_i does not authenticate id or if $|L| < k+1$, 1 otherwise. To this aim it executes the following steps:

- the service provider verifies the authenticity of the identity provider certificate $crt = \langle PK, k_{max}, \sigma_{crt}\rangle$ by verifying signature σ_{crt}. The result of this operation can be cached as long as the set PK is not refreshed;
- it verifies that $|L| \geq k+1$;
- it verifies all signatures in $\{\sigma_i\}$ by executing $\mathsf{Verify}(pk_i, id, \sigma_i)$;
- if any of the previous checks fails it outputs 0, otherwise it outputs 1.

$crt' \leftarrow \mathsf{Refresh}(sk_{IdP}, crt, PK_a, PK_r)$: the identity provider sets the new identity provider certificate crt', given the current identity provider certificate crt, the new set of public keys PK_a and the set of revoked public keys PK_r. To this aim he executes the following steps:

- the identity provider recovers all compromised identity servers and establishes the set of additional identity servers, if any.
- each additional or recovered identity server in the new set computes his signing key pair $\langle sk_i, pk_i\rangle$ by executing the $\mathsf{KeyGen}()$ algorithm, produces the corresponding certificate signing request π_i and sends it to the identity provider.
- the identity provider collects the set of new public keys PK_a, verifies the corresponding certificate signing requests $\{\pi_i\}_{i\in|PK_a|}$, and defines the set PK_r of public keys to revoke. PK_r includes the old public keys of recovered servers and any disposed server that is excluded from the protocol.
- the identity provider sets the new set of public keys $PK' = PK \cup PK_a \setminus PK_r$ such that $|PK'| = |PK|$, and authenticates the pair $\langle PK', k_{max}\rangle$ by signing it with his private key, producing $crt' = \langle PK', k_{max}, \sigma_{crt'}\rangle$ which is made publicly available.

6 Security guarantees

In this section we discuss the security of the token release scheme against attack classes A and C that we presented in Section 4. We devote Section 7 to consider class B attacks and evaluate the security of the overall SSO protocol when the token release scheme is integrated with different credential management systems. Here, we show how to mitigate class C attacks that violate the confidentiality of the identity provider signing key. Moreover, we show that the proposed token release scheme is secure against class A attacks that aim to violate the token release system within identity servers. Furthermore, we show that security against class A attacks holds during all violation patterns (Figure 2) that is, violation during: one operation period (1), one Refresh execution (2), consecutive operation periods (3) and consecutive Refresh executions (4).

A1. The attacker can violate the confidentiality of the token signing key of up to k identity servers through any of the violation patterns. The attacker can force a compromised identity server to execute the ReleasePartial operation on identities of the attacker's choice. We note that the attacker can obtain the same level of violation even if it only controls the key without knowing its value (e.g. the key is protected by an HSM). Attack A1 with violation pattern 1 is ineffective because an attacker that controls no more than k identity servers is not able to forge a valid authentication token. Attack A1 is ineffective even with the violation pattern 2. Here, the identity provider verifies that the certificate signing request submitted by identity servers is legitimate before certifying the new public keys. Moreover, an adversary that has access to the new secret keys produced during Refresh execution is unable to obtain a valid authentication token since he does not control enough $(k+1)$ identity servers. Finally, attack A1 is also ineffective with violation patterns 3 and 4. Given that identity servers are recovered at the beginning of the Refresh procedure, the adversary can never control more than k identity servers within any time period. Therefore, the adversary does not control enough $(k+1)$ identity servers to forge authentication tokens. Moreover, the public keys of the recovered identity servers are revoked during Refresh. We note that the attacker may try to exploit race conditions during revocation of public keys (e.g., CRL propagation delays) to force the service provider to verify authentication tokens with revoked public keys. To prevent this type of attacks we rely on the identity provider as an online certification authority. In this way the service provider always validates authentication tokens with a fresh copy of the new set of public keys PK'. As a result, the service provider can easily detect invalid authentication tokens authenticated by revoked public keys.

A2. Attack A2 is ineffective in any violation pattern because the proposed token release scheme allows the user to detect bogus partial tokens, discard them and repeat the *Sign-on* operation with another honest identity server; its existence is guaranteed by to the presence of an honest majority.

A3. Attack A3 is ineffective in any violation pattern because the Refresh operation ensures that at any moment there is an available honest majority of $k+1$ identity servers that is able to respond to users.

C1. The attacker may try to violate the identity provider signing key. However, we note the identity provider secret key sk_{IdP} can be kept offline as it must be used only during Setup and Refresh operations which occur on a much broader frequency than operations involving the identity servers signing keys. As a result, the signing key of identity provider can be protected to ensure it less vulnerable than the signing keys of identity servers. We remind that the survivability of the proposal could be extended to the identity provider by instantiating it as a collective entity as already discussed in Section 3.

The proposed protocol guarantees also the *accountability* security property that is, it allows an identity provider to attribute protocol deviations to specific compromised identity servers. This property is important because it allows an identity provider to prioritize recovery operations when servers are compromised, and can be complementary to already deployed approaches for monitoring system infrastructure [4]. All cryptographic material issued by identity servers is accountable. For example, each partial token σ_i produced during the ReleasePartial procedure is accountable because it can be verified through the public key of identity server i. Moreover, the authentication token $\langle id, \{\sigma_i\}, L\rangle$ computed during the Combine procedure is accountable because the set of signers L explicitly indicates the identity servers that contributed to signatures $\{\sigma_i\}$. It is important to note that related works [3,6,26] relying on threshold signatures are not completely accountable. Partial tokens computed by identity servers during the *Release partial* procedure may be accountable depending on the adopted scheme, whereas authentication tokens are not accountable. Other papers (e.g., [3,26]) adopt the threshold signature scheme of Boldyreva [9] which allows accountability of partial tokens because identity servers publish a commitment of their secret shares after the *Setup* and *Refresh* procedures. The work in [6] proposes an original RSA-based threshold signature scheme which blinds partial tokens thus preventing partial token accountability. The authentication token computed during the *Combine* procedure of [3,6,26] is not accountable because a key property of threshold signatures is that they do not reveal the identity of individual signers but only the cardinality of the set of signers [9].

The proposed original token release scheme guarantees several flexibility benefits: adjustable security threshold, adjustable performance overhead and compatibility with non-survivable SSO, that we discuss below. The proposed token release scheme guarantees an *adjustable security threshold* because the service provider can decide the value of k during the VerifyToken operation. This allows the choice of the best trade-off between security and performance for the service he offers, provided that $0 \leq k \leq k_{max} = \lfloor \frac{n}{2} \rfloor$. The constraint $k_{max} = \lfloor \frac{n}{2} \rfloor$ guarantees availability in case of k_{max} disconnected identity servers.

The scheme also guarantees an *adjustable performance overhead* because it allows the service provider to adjust the value of k to the best trade-off between performance and security. Higher values of k imply higher security in terms of token unforgeability and availability, as the adversary is required to violate $k+1$ identity servers to issue rogue authentication tokens or impede their release. However, these higher k values imply higher overheads because a user executes

the sign-on procedure with $k + 1$ servers. Previous proposals [3,6,26], which are based on threshold signatures, do not achieve a comparable flexibility because the value k is not decided by the service provider, but by the identity provider during the *Setup* operation. This value cannot be changed afterwards.

As a final attribute, the proposed token release scheme preserves *compatibility with non-survivable SSO*. If a service provider sets $k = 0$ during the VerifyToken operation, its users execute the SSO procedure with one identity server as in traditional non-survivable SSO.

7 Integration with credentials verification and storage protocols

We consider different credentials storage and verification protocols to evaluate their impact on the security of the SSO protocol. To this aim, we consider the attack category B described in Section 4, which involve credentials verification and storage protocols as follows:

- B1: integrity violation of credentials database;
- B2: unavailable identification protocol;
- B3: credentials database leak;
- B4: internal state leak;
- B5: Man-In-The-Middle (MITM) attacks from compromised identity servers.

We do not consider impractical credential storage and verification protocols that require users to maintain multiple credentials for identification, even if this is a naive solution to achieve survivable SSO.

We consider the following categories of authentication protocols that can be adopted to build a survivable SSO system:

- **P1**: a strawman approach based on plaintext storage and verification;
- **P2**: approaches that protect password storage but where verification is operated in plaintext [7];
- **P3**: approaches where passwords are protected at verification and storage time [18];
- **P4**: approaches that use secret sharing techniques to distribute passwords over multiple servers [3,6,26];

The presence of a majority of honest servers guarantees security against attacks B1 and B2 independently of the adopted authentication protocol and violation pattern. Even if the adversary registers malicious credentials (B1), he is unable to obtain enough valid partial tokens. Moreover, if the adversary prevents the completion of the authentication protocol in compromised identity servers (B2), the remaining honest majority ensures availability. If an authentication protocol is vulnerable to attacks B3 and B4, then the adversary can recover user credentials and impersonate him by compromising one identity server. Hence, violation patterns are not relevant to evaluate the security of protocols that are vulnerable to B3 and B4 because they already fail with one compromised sever.

P1. As a strawman approach, we consider a protocol in which each identity server stores and verifies plaintext passwords. The user sends the password over the secure channel to the identity server which verifies that it matches the stored password. This protocol yields a SSO protocol that is vulnerable to attacks B3, B4 and B5. The protocol is not survivable as an adversary that either compromises a single credentials database (B3) or observes the password verification procedure of one identity server (B4), can naively recover the password. The protocol is vulnerable to R5 (MITM) because, even if the communication channel is authenticated, the adversary may forward messages from compromised identity servers which are legitimate endpoints of the authenticated channel.

P2. Protocols that protect password storage but verify passwords in plaintext rely on password-hashing techniques [7]. Adopting these protocols in the considered distributed SSO scenario requires the user to transmit his password over a secure channel to each identity server which compares the password with its corresponding digest. This protocol is not completely secure against attack B3 because an adversary that captures the credentials database of any identity server can mount offline dictionary attacks (ODA). This may allow the attacker to recover the password to impersonate the user at the other identity servers. Moreover, the resulting SSO protocol is vulnerable to attacks B4 and B5 if an identity server is compromised. This protocol is vulnerable to B4 because a compromised identity server receiving the plaintext password can impersonate the user. It is also vulnerable to B5 because an adversary can forward the received plaintext password and impersonate the user to other identity servers.

P3. Security against server violation can be achieved through authentication protocols where identity servers never access plaintext passwords. The current state-of-the-art is represented by the OPAQUE scheme [18], which allows a user to store a secret key encrypted with his password at the registration phase. During authentication, only a user who knows the correct password can decrypt the secret key to prove his identity. The scheme adopts an oblivious PRF [14] which does not disclose any information about the password nor the secret key to the identity server during the registration and authentication phases. OPAQUE is not vulnerable to B5 because adopts channel bindings to prevent MITM attacks. While the SSO protocol obtained by different OPAQUE instances with each identity server is not vulnerable to B5, it is not completely secure against B3 and B4 because an adversary that captures the credentials database or observes the verification procedure of a single identity server may be able to mount offline dictionary attacks.

P4. We conclude by evaluating proposals that are secure against attacks B3, B4 and B5 relying on schemes based on secret sharing, such as Threshold Oblivious PRFs (TOPRF) [3,6,26]. The work of [3] is secure against these attacks considering static corruptions of identity servers, whereas [6,26] are secure in all violation patterns as they are proven secure against mobile adversaries. They are secure against B3 and B4 because they rely on techniques based on secret sharing to store and transmit the password. Hence, individual messages and credential databases do not contain enough information to mount attacks that can recover

the password, such as offline dictionary attacks. These proposals are also secure against B5 because they are proven secure against active adversaries that can eavesdrop communications of up to a threshold of other identity servers.

	B3 verification key	**B4** internal state	**B5** MITM
P1 - Strawman	○	○	○
P2 - Pwd Hashing	◑	○	○
P3 - OPAQUE	◑	◑	●
P4 - Secret sharing	●	●	●

○: vulnerable, ◑: partially vulnerable (offline dictionary attack), ●: not vulnerable

Table 1. Security guarantees of survivable single sign-on systems with different authentication protocols

The trade-offs of different authentication protocols are summarized in Table 1, where columns denote SSO requirements and rows denote the considered protocols. Password-based protocols that are not designed to be survivable, are either insecure against B3, or only partially secure. We note that when a protocol is partially secure against B3 due to possible offline dictionary attacks, the user can choose a strong password to make these attacks ineffective.

8 Conclusions

We propose the first flexible and survivable SSO protocol that relies on a distributed architecture of identity servers that collectively authenticate users and issue SSO tokens through a novel scheme. Flexibility allows service providers to choose the best trade-off between performance and security for each service and to preserve compatibility with non-survivable SSO. Survivability allows the identity provider to guarantee a high level of identity assurance even in presence of successful intrusions. We evaluate the security of the overall survivable SSO by considering several state of the art authentication protocols. Moreover, we show that the proposed token release scheme is secure against a comprehensive set of attack classes. Flexibility and survivability make the proposal a viable solution to offer secure and robust authentication to cloud services and any mission critical system that must rely on SSO. The results of this paper are open to different developments. It should be interesting to investigate how emerging passwordless authentication protocols may impact flexibility in the context of survivable SSO systems. Moreover, we think that it is possible to extend this proposal to support decentralized management systems as in the context of multi-cloud environments.

References

1. 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. Tech. rep., European Parliament, Council of the European Union (2014), https://eur-lex.europa.eu/eli/reg/2014/910/oj
2. Detecting abuse of authentication mechanisms. Tech. rep., National Security Agency (2020)
3. Agrawal, S., Miao, P., Mohassel, P., Mukherjee, P.: PASTA: PASsword-based Threshold Authentication. In: Proc. ACM SIGSAC Conf. Computer and Communications Security (2018)
4. Andreolini, M., Pietri, M., Tosi, S., Balboni, A.: Monitoring large cloud-based systems. In: 4th Int'l Conf. Cloud Computing and Services Science, CLOSER. SciTePress (2014)
5. Avizienis, A., Laprie, J.., Randell, B., Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing (2004)
6. Baum, C., Frederiksen, T.K., Hesse, J., Lehmann, A., Yanai, A.: PESTO: Proactively Secure Distributed Single Sign-On, or How to Trust a Hacked Server. 5th IEEE European Symp. Security and Privacy (2019)
7. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2: new generation of memory-hard functions for password hashing and other applications. In: European Symp. Security and Privacy. IEEE (2016)
8. Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S., Cooper, D.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Tech. rep., IETF (2008)
9. Boldyreva, A.: Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In: Int'l Work. Public Key Cryptography (2003)
10. Cash, D., Meltzer, M., Koessel, S., Adair, S., Lancaster, T.: Dark halo leverages solarwinds compromise to breach organizations. Tech. rep., Volexity (2020), https://www.volexity.com/blog/2020/12/14/dark-halo-leverages-solarwinds-compromise-to-breach-organizations/
11. Damgård, I., Jakobsen, T.P., Nielsen, J.B., Pagter, J.I.: Secure Key Management in the Cloud. In: IMA Int'l Conf. Cryptography and Coding. Springer (2013)
12. Eldefrawy, K., Ostrovsky, R., Park, S., Yung, M.: Proactive Secure Multiparty Computation with a Dishonest Majority. In: Int'l Conf. Security and Cryptography for Networks. Springer (2018)
13. Fisher, W., Grassi, P., Dog, S., Jha, S., Kim, W., McCorkill, T., Portner, J., Russell, M., Umarji, S., Barker, W.: Mobile Application Single Sign-On: Improving Authentication for Public Safety First Responders (2nd Draft). Tech. rep., National Institute of Standards and Technology (2019), Special Publication 1800-13B
14. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword Search and Oblivious Pseudorandom Functions. In: Theory of Cryptography Conference. Springer (2005)
15. Garcia, M., Bessani, A., Gashi, I., Neves, N., Obelheiro, R.: Analysis of operating system diversity for intrusion tolerance. Software: Practice and Experience (2014)
16. Grassi, P.A., Garcia, M.E., Fenton, J.L.: Digital identity guidelines. Tech. rep., National Institute of Standards and Technology (2017), DRAFT Special Publication 800-63c

17. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive Secret Sharing Or: How to Cope With Perpetual Leakage. In: Annual Int'l Cryptology Conference. Springer (1995)
18. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An Asymmetric PAKE ProtocolSecure Against Pre-Computation Attacks. In: Annual Int'l Conf. Theory and Applications of Cryptographic Techniques. Springer (2018)
19. Jhawar, R., Piuri, V.: Fault Tolerance and Resilience in Cloud Computing Environments. In: Computer and information security handbook. Elsevier (2017)
20. Magnanini, F., Ferretti, L., Colajanni, M.: Scalable, confidential and survivable software updates. IEEE Transactions on Parallel and Distributed Systems (2022). https://doi.org/10.1109/TPDS.2021.3090330
21. Nikitin, K., Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Gasser, L., Khoffi, I., Cappos, J., Ford, B.: CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In: Proc. 26th USENIX Security Symp. (2017)
22. Nystrom, M., Kaliski, B.: PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986 (2000), https://rfc-editor.org/rfc/rfc2986.txt
23. Ostrovsky, R., Yung, M.: How To Withstand Mobile Virus Attacks. In: Proc. Tenth Ann. ACM symp. Principles of distributed computing (1991)
24. Ray, I., Belyaev, K., Strizhov, M., Mulamba, D., Rajaram, M.: Secure logging as a service—delegating log management to the cloud. IEEE Systems Journal (2013)
25. Rose, S., Borchert, O., Mitchell, S., Connelly, S.: Zero Trust Architecture. Tech. rep., National Institute of Standards and Technology (2020)
26. Zhang, Y., Xu, C., Li, H., Yang, K., Cheng, N., Shen, X.S.: PROTECT: Efficient Password-Based Threshold Single-Sign-On Authentication for Mobile Users against Perpetual Leakage. IEEE Trans. Mobile Computing (2020)
27. Zhou, L., Schneider, F.B., Van Renesse, R.: COCA: A Secure Distributed On-line Certification Authority. ACM Transactions on Computer Systems (TOCS) (2002)