Fast IDentity Online with Anonymous Credentials (FIDO-AC)

Wei-Zhu Yeoh, *CISPA Helmholtz Center for Information Security;* Michal Kepkowski, *Macquarie University;* Gunnar Heide, *CISPA Helmholtz Center for Information Security;* Dali Kaafar, *Macquarie University;* Lucjan Hanzlik, *CISPA Helmholtz Center for Information Security*

https://www.usenix.org/conference/usenixsecurity23/presentation/yeoh

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Fast IDentity Online with Anonymous Credentials (FIDO-AC)

Wei-Zhu Yeoh[*]
CISPA Helmholtz Center for Information Security

Michal Kepkowski
Macquarie University

Gunnar Heide
CISPA Helmholtz Center for Information Security

Dali Kaafar
Macquarie University

Lucjan Hanzlik
CISPA Helmholtz Center for Information Security

Abstract

Web authentication is a critical component of today's Internet and the digital world we interact with. The FIDO2 protocol enables users to leverage common devices to easily authenticate to online services in both mobile and desktop environments, following the passwordless authentication approach based on cryptography and biometric verification. However, there is little to no connection between the authentication process and users' attributes. More specifically, the FIDO protocol does not specify methods that could be used to combine trusted attributes with the FIDO authentication process generically and allow users to disclose them to the relying party arbitrarily. In essence, applications requiring attributes verification (e.g., age or expiry date of a driver's license, etc.) still rely on ad-hoc approaches that do not satisfy the data minimization principle and do not allow the user to check the disclosed data. A primary recent example is the data breach on Singtel Optus, one of the major telecommunications providers in Australia, where very personal and sensitive data (e.g., passport numbers) were leaked. This paper introduces FIDO-AC, a novel framework that combines the FIDO2 authentication process with the user's digital and non-shareable identity. We show how to instantiate this framework using off-the-shelf FIDO tokens and any electronic identity document, e.g., the ICAO biometric passport (ePassport). We demonstrate the practicality of our approach by evaluating a prototype implementation of the FIDO-AC system.

1 Introduction

Web authentication is a crucial component of the digital world and the Internet we know today. The predominant web authentication method is via the login and password mechanism. The password is considered a single factor of authentication. In most modern applications, users are recommended to use multiple authentication factors. Such a factor could be an SMS code sent to the user's phone number or a designated mobile application requesting the user's acknowledgment.

The state-of-the-art solution is, however, based on cryptographic tokens. Those can store cryptographic keys and perform public key cryptography. The tokens, introduced by the Fast IDentity Online (FIDO) Alliance, are the most prominent instantiation of this idea, and together with the open-source FIDO2 protocol are a strong candidate for building advanced authentication frameworks. However, the main disadvantage of the current solution is that there is no link between the user's attributes and the authentication process, which limits the potential application space or forces the service to use ad-hoc solutions that are not bound to FIDO authentication.

The importance of attribute-based authentication is known in the research community, which has proposed many solutions. The most interesting ones are anonymous credentials, which allow users to disclose the attributes to service providers arbitrarily. Unfortunately, as time has shown, many of these approaches are not used in the real world and are far from what we consider practical. No tools and methodologies exist to efficiently combine anonymous credentials and attributes, in general, with the FIDO authentication process. The Meta Research group (formerly Facebook Research) reached the same conclusion . They issued a call for projects to develop such solutions [1]. Potential solutions to this problem would significantly influence how authentication systems are used and what we can use them for. One of the use cases is age verification, which is not a problem in most cases but becomes one if the service is legally obligated to check age (e.g., for selling alcohol or serving adult-only content). Even though those websites are required to verify the user's age, in practice, they ask the user to assert without further verification.

Those solutions are not limited to age verification but involve many practical systems where data minimization is needed but not implemented. This is evident due to many data breaches leaking the full data of identity documents. An interesting example is the recent data breach suffered by the

[*]We thank Saarland University for supporting Wei-Zhu Yeoh.

[1]https://research.facebook.com/research-awards/2022-privacy-enhancing-technologies-request-for-proposals/

Australian telecommunication company Optus [2], in which the hackers gained unauthorized access to two unique identity documents. The scope of this attack was significant because Optus did not employ techniques to minimize the personal data stored in the database, such as using attribute-based authentication that would allow the user to disclose only the minimal data required to receive the phone number.

Contributions. This paper introduces a novel approach for presenting claims in a privacy-preserving manner through a commercially recognized authentication protocol. To the best of our knowledge, our design is the first that presents an innovative and generic way to combine a privacy-enhancing technology (e.g., anonymous credentials and eID solutions) with a strong authentication protocol (i.e., FIDO2). Our approach to the framework definition is as follows. We introduce the building blocks of FIDO-AC and define the requirements for the system. Then, we formally define the notion of passwordless authentication with attributes, which we later extend with a mediator party to meet our requirements. Our security models can be of independent interest and used as a foundation for securely integrating attributes into the FIDO standard. In the next step, we introduce the system design and formal protocol flow, which we enrich with a security and threat analysis. Finally, we present our implementation of the FIDO-AC system and its performance evaluation. Notably, the design of our system recognizes and addresses the well-known challenges in integrating the existing deployments. Therefore, we believe that FIDO-AC can be effortlessly used in any commercial solution to elevate the privacy of Personal Identifiable Information (PII). To summarize, our contributions are as follows:

- **FIDO-AC framework.** We propose a complete and industry-ready solution for utilizing anonymous credentials with local or remote attestation through a FIDO2 channel.
- **FIDO2 Extension.** We introduce a new FIDO2 extension and a mechanism to bind the extension data with the FIDO2 assertion in the constrained environment of the WebAuthn API implementations.
- **System evaluation.** We provide a comprehensive evaluation of the FIDO-AC framework. We discuss the security and privacy properties, as well as the usability from the user's and relying party's points of view.
- **Implementation.** We prove the feasibility of our design by developing and openly publishing a prototype implementation.

2 Background and Related Work

This section will briefly explain the FIDO2 standard and how the authentication process works. Later, we will describe electronic identity documents (focusing on ePassport) and anonymous credentials. The abbreviations FIDO and FIDO2 are used interchangeably, as they refer to the same protocol.

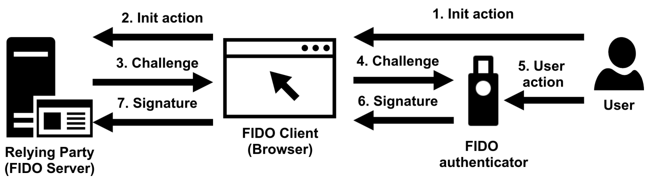

Figure 1: FIDO2 parties and simplified authentication flow

2.1 FIDO2

FIDO2 is an authentication protocol designed by FIDO Alliance in collaboration with vendors and identity and access management (IAM) experts. The open-source nature and broad support (i.e., all major web browsers and OSes) make FIDO2 a serious candidate for becoming the de facto standard for second-factor and passwordless authentication.

The FIDO standard defines two processes (ceremonies): registration and assertion. The former allows the creation of a link between the server and the authenticator. The latter is used to prove the authenticator's possession (e.g., a cryptographic token). Both are built on a simple request-response transaction that generates verifiable proof. Usually, three parties participate in the flow: FIDO Server, FIDO Client, and the authenticator. As presented in Figure 1, the authentication flow starts with a trigger sent to the FIDO Server (steps 1. and 2.). The trigger might be an automated action or user interaction (e.g., the user clicks the login button). Then, the FIDO Server generates a random challenge that travels through the FIDO Client to the authenticator (steps 3. and 4.). In WebAuthn, the user's action is required to unlock the authenticator (step 5.). Finally, the authenticator generates a signature (using the preregistered key) and sends it back to the FIDO Server (steps 6. and 7.). Interestingly, the transportation layer of FIDO2 is composed of two related protocols: CTAP [16] and WebAuthn [35]. The former is responsible for communication with an authenticator (i.e., binary messages sent via USB, BLE, or NFC channels), whereas the latter describes the API for the client side.

The design of FIDO2 follows the *"privacy by design"* principle. Unlike other methods that require Personal Identifiable Information (PII) to function (e.g., phone number in SMS OTP case), FIDO2 does not need any PII. Furthermore, FIDO2 ensures that the protocol does not compromise privacy (e.g., by linking accounts using key handles). Both registration and authentication incorporate privacy-preserving mechanisms. A new key pair with randomly-looking key handles is generated for each registration. The authenticator identifier (AAGUID) and attestation mechanism guarantee that the authenticator cannot be uniquely identified. In the case of authentication, each transaction uses a random challenge.

2.2 Electronic Identity Documents

Electronic identity documents (eIDs) are standard documents with an electronic layer capable of storing data and executing cryptographic protocols. The most widely used eID is the biometric passport (ePassport) introduced by the International Civil Aviation Organization (ICAO) and issued in more than 150 countries [34]. According to EU regulation, 2019/1157 [28], all European Union members must include an application supporting the ICAO in their national identity documents, making this the de facto standard for eIDs. Below we will give a high-level overview of the cryptographic protocols included in the ICAO standard.

Basic Access Control (BAC) is a password-based mechanism designed to thwart both online skimming and offline eavesdropping attacks. Attackers without the knowledge of the password (document number, date of birth, and expiry date) will be unable to read the passport's content and decipher the eavesdropped communication. However, the security of the BAC suffers from offline dictionary attacks due to the low entropy of the password. Its successor, Password Authentication Connection Establishment (PACE), provides a much better security guarantee by employing a Diffie-Hellman key.

Passive Authentication (PA) enables the reader to verify the authenticity of the eID data. During PA, the reader will retrieve, in addition to the data, the document security object (DSO), which encompasses a signature on the hash values of the data. The reader can then verify the authenticity of the data by comparing the hashed values and verifying the signature using the issuer's public key. It is worth noting that the data is stored in so-called data groups (DG). This data contains personal data and random numbers (e.g., eID number). Moreover, the hash value of the data groups for the same personal data will be different, i.e., we assume that $\mathsf{H}(DG_1)$ and $\mathsf{H}(DG_2)$ are unlinkable, if only knowing the personal data and not the random numbers.

Chip Authentication (CA) prevents the cloning of eID and its data. Once read, the data could potentially be uploaded to a fresh eID, practically cloning the original eID. The problem is solved via a hardware assumption. We assume that the secret key loaded to the eID during the personalization phase cannot be extracted. The corresponding public key is added to the signed data, creating a link between it and the device. During verification, the reader checks that the public key used by the eID during this additional step is part of the data verified during passive authentication.

2.3 Anonymous Credentials

Anonymous credential systems (ACs) are a cryptographic building block envisioned and introduced by Chaum [9]. They allow users to obtain digital credentials encoding their attributes from an issuer. Users can later use those credentials to prove certain claims (e.g., over 18 years old) without revealing any other meaningful information about themselves.

Anonymous credentials have found applications in various problems and environments. Those include keyed-verification ACs [8, 13], AC as delegated parts of the credential to other parties [4, 6, 14], AC in the decentralized [17, 33] or cloud-based [24] setting. An interesting system, which can be treated as a single-use, single-attribute credential, is Privacy Pass [15]. It was introduced as a way to solve the CAPTCHA problem that anonymous network users are facing. There also exists a rate-limiting version [22] (introduced in iOS 16 [2]), which uses a trusted mediator to enforce the issuer's policy. The privacy guarantees are that the mediator does not learn the origin where the user will redeem the token. At the same time, the issuer is oblivious to any data identifying the user, e.g., IP address or other distinguishing metadata. The exciting part of this paper is that in this setting, secure hardware components of the iPhone, together with iCloud, play the role of the trusted mediator. We will later see that the architecture of our solution is similar to this.

Recently, Rosenberg et al. [31] introduced the idea of using the blockchain consensus to make anonymous credential issuers obsolete. The idea is to use an existing identity infrastructure and store the credentials in a secure data structure. They use a bulletin board based on Ethereum smart contracts for storage in their implementation. The exciting part related to our work is that they focus on using the existing infrastructure for eIDs, particularly the ICAO-based ePassport. They use the fact that the personal data stored on the ePassport is authenticated by a governmental authority via passive authentication. Unfortunately, their solutions fail to provide an active authentication of the ePassport. In particular, knowing the data stored on the eID and the DSO is enough to create the credentials in their system. However, this data is fully read during border control, making their system unusable.

Most notable in the context of our paper is the very recent work by Schwarz et al. [32] in which the trusted execution environment (TEE) is used for the AC. The authors propose FeIDo, a TEE-based roaming FIDO authenticator that computes FIDO credentials based on user attributes. Their main goal is to solve the token loss problem since the same keys can be accessed using a different eID because it stores the same attributes. Interestingly, they notice that since the TEE gets access to the user's data, it can enforce access policies. Our approach is very different. Firstly, we support any FIDO token and are not bound to a custom instantiation of the token. Secondly, we are not bound to only electronic identity documents but also support standalone anonymous credential systems or any other, e.g., cloud-based identity systems with the non-shareability property. Lastly and most importantly, in our approach, the personal data never leaves the user's device, which is not true for [32].

[2] https://developer.apple.com/videos/play/wwdc2022/10077

3 Requirements and Threat Model

The main objective of FIDO-AC is to provide a practical system capable of augmenting the FIDO2 protocol with anonymous credentials derived from a verifiable source (i.e., eID). The system guarantees that the data was gathered from a legitimate document at the time of a FIDO transaction, and only selected information about the user is shared with relying party. The requirements are crafted with the criteria listed below:

R.1 **Privacy Preserving.** At the end of the FIDO-AC protocol, the relying party should learn only the relevant authenticated user information without compromising user privacy. In particular, FIDO2 privacy guarantees should not be violated.

R.2 **Active Authentication (Liveliness).** The FIDO-AC system should verify the possession of a non-sharable credential/device for the presented user attributes.

R.3 **Compatibility.** FIDO-AC should be fully compatible with the FIDO2 protocol.

R.4 **User-Centric Design.** The solution should impose minimal user friction to ease the adoption of FIDO-AC.

R.5 **Pluggable Integration.** The integration of the FIDO-AC system with an existing FIDO2 deployment should be effortless in terms of development, operation, and deployment. In particular, FIDO-AC should work without modification of existing FIDO clients and authenticators.

R.6 **Efficient architecture.** The FIDO-AC system should deliver reasonable performance (compared to the pure FIDO system), and it should be trivial to scale. Moreover, the architecture should be vendor agnostic.

We design FIDO-AC to be an extension of the existing FIDO2 standard with the extra capability of providing anonymous credentials. The threat model of FIDO2 web authentication [1] is used and extended to include new elements unique to the proposed system. Following the original FIDO trust assumptions, we assume authenticated communication channels between different parties. The integrity of the client agent, browser, authenticator, and OS is trusted, which is the same trust assumption needed for FIDO authentication. Additionally, to accommodate the introduction of new modules, we trust the mediator party to perform the verification correctly and not collude with relying parties to link users. Moreover, we trust the eID infrastructure and the hardware-based protection of the eID device.

We assume the integrity of the underlying device hardware is correct and trusted. Side-channel attacks such as fault injection, power analysis, and micro-architecture side-channel attacks are also out-of-scope in this paper. We also do not consider denial-of-service (DoS) attacks on the mediator. Last but not least, we assume the FIDO-AC mobile applications and services are free of software vulnerability.

4 Passwordless Authentication with Attributes

In this section, we introduce passwordless authentication with attributes (PAwA). The passwordless authentication protocol (PLA) captures the syntax of WebAuth and was formally introduced and analyzed in [3] and subsequently in [21]. However, the PLA protocol does not capture the notion of attributes, which is needed for FIDO-AC. Therefore, we extend the definitions from [21] to formally introduce attributes to the passwordless authentication.

In other words, we formally model how to use attributes in the FIDO framework. Unfortunately, it turns out that achieving this model with existing FIDO and credential systems while simultaneously fulfilling the requirements defined in the previous section is hard. Therefore, we will further extend this model by introducing an additional trusted party to the system called a mediator. This new party will act as a sort of interface between FIDO and the credential system. It will also introduce new privacy concerns, which will be addressed by formally introducing the mediator into our definitions of PAwA.

4.1 Formal Model of PAwA

The PAwA protocol consists of two processes, namely the registration phase and the authentication phase. The messages passed between a server and a token are relayed through an intermediary client interface, e.g., the web browser. In both phases, the server sends the first message containing a challenge and the desired attribute policy. We follow the same syntax as the one used in [3, 21] and encapsulate the policy as part of the server's challenge [3]. After receiving the challenge, the client, together with the token, computes the response, similar to standard PLA. Depending on the instantiation of PAwA, the client can either forward the attribute policy to the token or compute this part of the assertion locally.

In the registration phase, the token additionally attaches a public key pk and a credential identifier cid. The server keeps track of the information received from the token in its storage. Then, in subsequent authentication, the server includes the cid in the first message described above. In PAwA, the server verifies the signed message with respect to the augmented challenge. The server also verifies the response with respect to its attribute policy.

4.1.1 Formal Syntax

The model considers parties $\mathcal{P} = \mathcal{T} \cup \mathcal{S}$, where the parties are partitioned into the set of tokens $\mathcal{T}$ and the set of servers $\mathcal{S}$. Each token $T \in \mathcal{T}$ has a fixed state that is initialized with a key msk_T. Additionally, we associate an attribute set Att_T with each token T, where attributes are elements from a set Att_U.

[3]Note that this is in line with how one would implement the attribute policies as part of the extension fields of a FIDO challenge.

Each server $S \in \mathcal{S}$ constructs a key-value table known as the registration context rcs_S, whereby a new entry will be inserted whenever a token registers with the server. Each server is also uniquely identified with its publicly known unique identifier id_S, which in practice, corresponds to a URL. The server is also assumed to know the user account and its cid. Moreover, each server specifies an access policy $\mathsf{Policy}_S \subseteq \mathsf{Att}_\mathsf{U}$.

The syntax of the PAwA protocol is formally defined in Definition 1. To model the capability of the adversary to freely communicate with tokens and servers, a server oracle and a token oracle are additionally defined in Definition 3, whereby st_S is used to model the state transfer between algorithms, C_s is used to bind registration to authentication, and the $\pi^{i,j}$ handle is used to model the instances of registration and authentication. Partnering of two handles $\pi_S^{i,j}$ and $\pi_T^{i',j'}$ as defined in Definition 4, for which they share the same session identifier, is used as the winning condition of the security experiments.

Definition 1 (PAwA)**.** A passwordless authentication scheme with attributes (PAwA) is a tuple $\mathsf{PAwA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$:

- Gen: on input parameters par, outputs a secret key msk.
- Reg: given as a tuple of the following algorithms:
  - $\mathsf{rchall}_\mathsf{ac}$**:** on input of a server identity id_S, outputs challenge with policy value c_p and a state st.
  - $\mathsf{rcomm}_\mathsf{ac}$**:** on input of a server identity id_S and a challenge c_p, outputs a message M_r.
  - $\mathsf{rresp}_\mathsf{ac}$**:** on input of a master secret key msk, a server identity id_S and a message M_r, outputs credential identifier cid and a response R_r.
  - $\mathsf{rcheck}_\mathsf{ac}$**:** on input of a state st, a credential identifier cid and a response R_{ac}[4], outputs a bit b and a credential cred.
- Auth: given as a tuple of the following algorithms:
  - $\mathsf{achall}_\mathsf{ac}$**:** on input of a server identity id_S, outputs a challenge c_p and a state st.
  - $\mathsf{acomm}_\mathsf{ac}$**:** on input of a server identity id_S and a challenge c_p, outputs a message M_a.
  - $\mathsf{aresp}_\mathsf{ac}$**:** on input of a master secret key msk, a server identity id_S, a credential identifier cid, and a message M_a, outputs a response R_a.
  - $\mathsf{acheck}_\mathsf{ac}$**:** on input of a state st, a registration context rcs, a credential identifier cid and a response R_{ac}[4], outputs a bit $b \in \{0, 1\}$

Algorithms $\mathsf{rchall}_\mathsf{ac}$, $\mathsf{rcheck}_\mathsf{ac}$, $\mathsf{achall}_\mathsf{ac}$, $\mathsf{acheck}_\mathsf{ac}$ are executed by servers, $\mathsf{rcomm}_\mathsf{ac}$, $\mathsf{acomm}_\mathsf{ac}$ are executed by clients, and $\mathsf{rresp}_\mathsf{ac}$, $\mathsf{aresp}_\mathsf{ac}$ are executed by tokens.

Definition 2 (Policy Extraction)**.** We assume that there exists a function $\mathsf{Pol}-\mathsf{Ext}(M)$ that on input of the message outputs of $\mathsf{acomm}_\mathsf{ac}(\mathsf{id}_S, \cdot)$, and $\mathsf{rcomm}_\mathsf{ac}(\mathsf{id}_S, \cdot)$ returns the policy Policy_S.

[4]Depending on the flow type, authentication or registration, R_{ac} contains either R_a or R_r

Definition 3 (Server and Token Oracles)**.** Let $\mathcal{A}$ be an adversary algorithm and $\mathsf{PAwA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ be a passwordless authentication with attributes scheme. Each party $P \in \mathcal{T} \cup \mathcal{S}$ is associated with a set of handles $\pi_P^{i,j}$ that models two types of instances corresponding to registration and authentication. Each party is represented by a number of these instances. Concretely, $\pi_P^{i,j}$ for $j = 0$ is the i-th registration instance of party P and for $j \geq 1$ is the j-th authentication instance of P corresponding to the i-th registration.

It is assumed that for each token $T \in \mathcal{T}$, a secret key is generated as $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. For each server $S \in \mathcal{S}$, key-value tables $\mathsf{rcs}_S, C_S, \mathsf{st}_S$ are given. By default, these are empty. Adversary, $\mathcal{A}$ has access to oracles $\mathsf{Setup}, \mathsf{Start}, \mathsf{Challenge}, \mathsf{Complete}$ defined as follows:

- $\mathsf{Setup}(\mathsf{Policy}_{LS}, \mathsf{Att}_{LT})$: Executes $(\mathsf{Policy}_{S_1}, ..., \mathsf{Policy}_{S_n}) := \mathsf{Policy}_{LS}$, and $(\mathsf{Att}_{T_1}, ..., \mathsf{Att}_{T_m}) := \mathsf{Att}_{LT}$.
- $\mathsf{Start}(\pi_S^{i,j})$: This executes $(c_p, \mathsf{st}) \leftarrow \mathsf{rchall}_\mathsf{ac}(\mathsf{id}_S)$ in case $j = 0$ or $(c_p, \mathsf{st}) \leftarrow \mathsf{achall}_\mathsf{ac}(\mathsf{id}_S)$ in case $j > 0$. The oracle sets $\mathsf{st}_S[i, j] := \mathsf{st}$ and returns c_p to $\mathcal{A}$.
- $\mathsf{Challenge}(\pi_T^{i,j}, \mathsf{id}_S, \mathsf{cid}, M)$: Executes $(\mathsf{cid}, R_r) \leftarrow \mathsf{rresp}_\mathsf{ac}(\mathsf{msk}_T, \mathsf{id}_S, M)$ if $j = 0$ or $R_a \leftarrow \mathsf{aresp}_\mathsf{ac}(\mathsf{msk}_T, \mathsf{id}_S, \mathsf{cid}, M)$ if $j > 0$. $\mathcal{A}$ is given $((\mathsf{cid}, R_r)$ or $R_a)$.
- $\mathsf{Complete}(\pi_S^{i,j}, \mathsf{cid}, R)$: Aborts if $\mathsf{Start}(\pi_S^{i,j})$ has not been queried before. If $j = 0$, it executes $(b, \mathsf{cred}) \leftarrow \mathsf{rcheck}_\mathsf{ac}(\mathsf{st}_S[i, j], \mathsf{cid}, R)$, sets $C_S[i] := \mathsf{cid}$, and $\mathsf{rcs}_S[\mathsf{cid}] := \mathsf{cred}$. If $j > 0$, it aborts if $\mathsf{cid} \neq C_S[i]$. Otherwise, it executes $b \leftarrow \mathsf{acheck}_\mathsf{ac}(\mathsf{st}_S[i, j], \mathsf{rcs}_S, \mathsf{cid}, R)$. In both cases, b is returned to $\mathcal{A}$.

It is assumed that for each $(i, j, T, S) \in \mathbb{N} \times \mathbb{N} \times \mathcal{T} \times \mathcal{S}$, the oracles $\mathsf{Setup}(\cdot, \cdot)$, $\mathsf{Start}(\pi_S^{i,j})$, $\mathsf{Challenge}(\pi_T^{i,j}, \cdot, \cdot, \cdot)$, and $\mathsf{Complete}(\pi_S^{i,j}, \cdot, \cdot)$ are executed only once.

Definition 4 (Session Identifiers and Partnering)**.** Consider the oracles from Definition 3. Let V_t be a function that takes as input the transcript $tr_T^{i,j} = (\mathsf{id}_S, \mathsf{cid}, M, R)$ that a token $T \in \mathcal{T}$ observes in an oracle call to $\mathsf{Challenge}(\pi_T^{i,j}, \cdot, \cdot, \cdot)$, and outputs a bitstring $V_t(tr_T^{i,j})$. Similarly, let V_s be a function that takes as input the transcript $tr_S^{i,j} = (c, \mathsf{cid}, R)$ that a server $S \in \mathcal{S}$ observes in oracle calls to $\mathsf{Start}(\pi_S^{i,j}), \mathsf{Complete}(\pi_S^{i,j}, \cdot, \cdot)$, and outputs a bitstring $V_s(tr_S^{i,j})$. It is assumed that these functions are specified by PAwA. The handles $\pi_T^{i,j}$ and $\pi_S^{i',j'}$ are partnered if: $(j = 0 \iff j' = 0) \wedge V_t(tr_T^{i,j}) = V_s(tr_S^{i',j'})$.

4.1.2 Security and Privacy

We will now define what it means for a PAwA protocol to be secure. We begin with security against impersonation, which informally ensures that there is precisely one partnered session for an accepting server. The security is defined in Definition 5, for which the adversary can interact with tokens and servers concurrently by using the oracles defined in Definition 3.

Definition 5 (Impersonation Security (Adapted from [21])). For a $\mathsf{PAwA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ scheme, the following security experiment $\mathbf{Imp}^{\mathcal{A}}_{\mathsf{PAwA}}$ is defined to run between the challenger and an adversary $\mathcal{A}$.

- **Setup.** For each token $T \in \mathcal{T}$, a key is generated by running $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. The adversary assigns the attribute set for tokens and policy set for servers by calling the oracle Setup and passing in the attribute set lists.
- **Online Phase.** The adversary is allowed to interact with the oracles $\mathsf{Start}, \mathsf{Challenge}, \mathsf{Complete}$ as in Definition 3.
- **Output Phase.** Finally, $\mathcal{A}$ terminates, and the experiment outputs 1 if and only if there exists a server handle $\pi^{i,j}_S$ for $j > 0$ such that the following conditions hold:
  1. $\pi^{i,0}_S$ is partnered with a token handle $\pi^{k,0}_T$.
  2. $\pi^{i,j}_S$ accepted, i.e. in call $\mathsf{Complete}(\pi^{i,j}_S, \mathsf{cid}, R)$, algorithm $\mathsf{acheck}(\mathsf{st}_S[i,j], \mathsf{rcs}_S, \mathsf{cid}, R)$ returned 1.
  3. $\pi^{i,j}_S$ is not partnered with any token handle $\pi^{i',j'}_T$, or it is partnered with a token handle, which is partnered with a different server handle $\pi^{i'',j''}_{S'}$.

In addition to the standard FIDO security defined above, we will now define attribute unforgeability. Informally, we want to ensure that an adversary can only access the server if it possesses a token that adheres to the server's policy. We achieve this by requiring that if S accepts, then the partnered token must have the attribute set that satisfies the server policy.

Definition 6 (Attribute Unforgeability). For a $\mathsf{PAwA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ scheme, the following security experiment $\mathbf{Att\text{-}Unf}^{\mathcal{A}}_{\mathsf{PAwA}}$ is defined to run between the challenger and an adversary $\mathcal{A}$.

- **Setup.** For each token $T \in \mathcal{T}$, a key is generated by running $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. The adversary assigns the attribute set for tokens and policy set for servers by calling the oracle Setup and passing in the attribute set lists.
- **Online Phase.** The adversary is allowed to interact with the oracles $\mathsf{Start}, \mathsf{Challenge}, \mathsf{Complete}$ as in Definition 3.
- **Output Phase.** Finally, $\mathcal{A}$ terminates, and the experiment outputs 1 if and only if there exists a server handle $\pi^{i,j}_S$ for $j > 0$ such that the following conditions hold:
  1. $\pi^{i,0}_S$ is partnered with a token handle $\pi^{k,0}_T$.
  2. $\pi^{i,j}_S$ accepted, i.e., in call $\mathsf{Complete}(\pi^{i,j}_S, \mathsf{cid}, R)$, algorithm $\mathsf{acheck}(\mathsf{st}_S[i,j], \mathsf{rcs}_S, \mathsf{cid}, R)$ returned 1.
  3. $\pi^{i,j}_S$ is not partnered with any token handle $\pi^{i',j'}_T$, or it is partnered with a token handle for which its attribute set Att_T does not satisfy the server policy Policy_S.

Similarly to the standard FIDO security model, we introduce an unlinkability definition that ensures that tokens are not linkable across origins. We extend the unlinkability proposed in [21] to capture attributes. Informally, we ensure that the server cannot learn more attributes than its policy has requested. In the unlinkability experiment, the adversary $\mathcal{A}$ is given access to the oracles defined in Definition 7, and with it, $\mathcal{A}$ gains the global view of the system. $\mathcal{A}$ is given two additional oracles Left and Right, which run T_b and T_{1-b} for a random bit b. $\mathcal{A}$ is said to win the game if it can determine which token is used in which oracle with respect to the conditions of instance freshness and credential separation defined in Definition 3. Credential separation models attack that a server can launch when the same token is used twice at the same server, while instance freshness is a consequence of the oracles definition.

Definition 7 (Unlinkability (Adapted from [21])). For a $\mathsf{PAwA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$, following experiment $\mathbf{Unl}^{\mathcal{A}}_{\mathsf{PAwA}}$ is defined to run between the challenger and an adversary $\mathcal{A}$.

- **Setup.** For each token $T \in \mathcal{T}$, a key is generated by running $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. The adversary assigns the attribute set for tokens and policy set for servers by calling the oracle Setup and passing in the attribute set lists.
- **Phase 1.** The adversary is allowed to interact with oracles $\mathsf{Start}, \mathsf{Challenge}, \mathsf{Complete}$ (see Definition 3). Moreover, we allow the adversary to query the Challenge oracle in a way that the oracle also executes the client part of the execution, i.e., the Challenge oracle additionally executes the rcomm_{ac} or acomm_{ac} algorithms.
- **Phase 2.** The adversary outputs two (not necessarily distinct) token identifiers T_0, T_1, and two (not necessarily distinct) server identifiers $S_L, S_R \in \mathcal{S}$ such that:
  $$\mathsf{Att}_{T_0} \supseteq \mathsf{Policy}_S \iff \mathsf{Att}_{T_1} \supseteq \mathsf{Policy}_S,$$
  for all $S \in \{S_L, S_R\}$. Let i_0 and i_1 be the smallest identifiers for which the token handles $\pi^{i_0,0}_{T_0}$ and $\pi^{i_1,0}_{T_1}$ were not queried to the Challenge oracle in Phase 1. The experiment chooses a bit b uniformly at random. It sets $j_0 := 0, j_1 := 0$ and initializes two oracles $\mathsf{Left}, \mathsf{Right}$ as follows:
  - $\mathsf{Left}(\mathsf{cid}, M)$: Abort if $\mathsf{Pol-Ext}(M) \neq \mathsf{Policy}_{S_L}$, else return $\mathsf{Challenge}(\pi^{i_b,j_b}_{T_b}, \mathsf{id}_{S_L}, \mathsf{cid}, M)$ and set $j_b = j_b + 1$.
  - $\mathsf{Right}(\mathsf{cid}, M)$: Abort if $\mathsf{Pol-Ext}(M) \neq \mathsf{Policy}_{S_R}$, else return $\mathsf{Challenge}(\pi^{i_{1-b},j_{1-b}}_{T_{1-b}}, \mathsf{id}_{S_R}, \mathsf{cid}, M)$ and set $j_{1-b} = j_{1-b} + 1$.

  Like in Phase 1, we allow the adversary to decide if the Left and Right oracles should execute the client part algorithms.
- **Phase 3.** The adversary can interact with all the oracles defined in Phases 1 and 2.
- **Output Phase.** Finally, the adversary outputs a bit $\hat{b}$. Consider the following lists of cid's:
  - $\mathcal{L}^r_{ch}$ contains all cid's returned by queries that are not issued via $\mathsf{Left}, \mathsf{Right}$ and are of the form $\mathsf{Challenge}(\pi^{i,0}_T, \mathsf{id}_S, \cdot, \cdot)$ for any $i, T \in \{T_0, T_1\}$ and $S \in \{S_L, S_R\}$.
  - $\mathcal{L}^a_{ch}$ contains all cid's that are part of the input of queries that are not issued via $\mathsf{Left}, \mathsf{Right}$ and are of the form $\mathsf{Challenge}(\pi^{i,j}_T, \mathsf{id}_S, \cdot, \cdot)$ for any $j > 0, i, T \in \{T_0, T_1\}$ and $S \in \{S_L, S_R\}$.

– $\mathcal{L}^r_{lr}$ contains all cid's returned by queries to Left or Right when $j_b = 0$ or $j_{1-b} = 0$, respectively.

– $\mathcal{L}^a_{lr}$ contains all cid's that are part of the queries to Left or Right when $j_b > 0$ or $j_{1-b} > 0$, respectively.

The experiment returns 1 if and only if:
– bit $\hat{b}$ is equal to bit b, and
– (instance freshness) the adversary never made a query to oracle Challenge using handles $\pi^{i_0,k_0}_{T_0}$ and $\pi^{i_1,k_1}_{T_1}$ for any k_0, k_1, and
– (credential separation) The following set is empty:

$$\begin{aligned}\textbf{wUnl}: &\ (\mathcal{L}^r_{ch} \cup \mathcal{L}^a_{ch}) \cap (\mathcal{L}^r_{lr} \cup \mathcal{L}^a_{lr})\\ \textbf{mUnl}: &\ ((\mathcal{L}^r_{ch} \cup \mathcal{L}^a_{ch}) \cap \mathcal{L}^a_{lr}) \cup ((\mathcal{L}^r_{lr} \cup \mathcal{L}^a_{lr}) \cap \mathcal{L}^a_{ch})\\ \textbf{sUnl}: &\ (\mathcal{L}^r_{ch} \cap \mathcal{L}^a_{lr}) \cup (\mathcal{L}^r_{lr} \cap \mathcal{L}^a_{ch}) .\end{aligned}$$

Depending on credential separation, we distinguish three levels of unlinkability: weak, medium and strong.

4.2 Formal Model of PAwAM

In the previous section, we introduced a security model for FIDO with attributes. Unfortunately, introducing attributes without significant changes to the FIDO specification and token firmware is impossible, which violates our compatibility requirement R.3 and pluggable integration requirement R.5.

Our goal is to build a system that uses existing building blocks. In particular, we want to interface existing FIDO solutions with attribute-based systems (e.g., anonymous credentials or ICAO eID-based attributes). To this end, we must introduce a trusted party called a mediator that acts as the interface between both systems.

4.2.1 Formal Syntax

The mediator is identifiable using a public key pk_M with a corresponding secret key sk_M. This new party introduces additional security problems, which we capture formally in the definitions below. In our syntax, we will also use $\mathsf{ask}_T \leftarrow \mathsf{IssCred}(\mathsf{Att}_T)$ to denote a token-specific secret key for the attribute-based credential systems. To simplify our considerations, we abstract the attribute-based system issuing process using algorithm IssCred. We assume this algorithm outputs a fresh ask for the given attributes, leading to fresh credentials for the attribute-based system (e.g., new data groups in case of eID systems). It is worth noting that in PAwA implementations, the ask_T can be part of the token's master secret key. We make this key explicit in PAwAM to simplify the description. The user platform (i.e., token and client) can use this key to prove possession of attributes to the mediator. We assume that this key implicitly defines the attributes Att_T corresponding to token T.

Definition 8 (PAwAM)**.** A passwordless authentication scheme with attributes and mediators (PAwAM) is a tuple $\mathsf{PAwAM} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth}, \mathsf{Med})$:

• $\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth}$: has the same description as in PAwA.

• Med: given as a tuple of the following algorithms:

$\mathsf{attestreq}_{\mathsf{ac}}$**:** on input of a secret key ask_T, a server challenge c, outputs an attestation request req_M and nonce.
$\mathsf{attestchal}_{\mathsf{ac}}$**:** on input of a request req_M, a mediator secret key sk_M, a mediator public key pk_M, outputs an attestation state $\mathsf{st}_{\mathsf{chal}}$ and a challenge chal_M.
$\mathsf{attestresp}_{\mathsf{ac}}$**:** on input of an attribute secret key ask_T, a challenge chal_M, outputs an attestation response resp_M.
$\mathsf{attest}_{\mathsf{ac}}$**:** on input of a challenge state $\mathsf{st}_{\mathsf{chal}}$, a response resp_M and a mediator secret key sk_M, outputs a attestation message att_m and a signature σ_m.
$\mathsf{prove}_{\mathsf{ac}}$**:** on input of an attestation message att_m, an attestation signature σ_m, a nonce nonce, attributes Att, a policy Policy_S, outputs a proof of attribute possession Π_{Att}.
$\mathsf{check}_{\mathsf{ac}}$**:** on input of a proof Π_{Att}, a policy Policy_S, a mediator public key pk_M, and a challenge c, outputs a bit b_{ac}.

Algorithms $\mathsf{check}_{\mathsf{ac}}$ is executed by servers during the execution of $\mathsf{rcheck}_{\mathsf{ac}}$ and $\mathsf{acheck}_{\mathsf{ac}}$. Algorithms $\mathsf{attestchal}_{\mathsf{ac}}$ and $\mathsf{attest}_{\mathsf{ac}}$ are executed by mediators, $\mathsf{attestreq}_{\mathsf{ac}}$, $\mathsf{attestresp}_{\mathsf{ac}}$ and $\mathsf{prove}_{\mathsf{ac}}$ are executed by clients.

Definition 9 (Oracles)**.** For PAwAM, we use the same server and token oracle defined for PAwA. We slightly modify the Setup oracle, which now additionally sets the keys ask_T of tokens according to the attributes the oracle sets. Additionally, we allow the adversary to communicate with the mediator M^i, where i represents the i-th session of the mediator using the following oracles:
• $\mathsf{MedReq}(T, c)$: The oracle executes $(\mathsf{nonce}, \mathsf{req}_M) \leftarrow \mathsf{attestreq}_{\mathsf{ac}}(\mathsf{ask}_T, c)$. The result is returned to $\mathcal{A}$.
• $\mathsf{MedChal}(M^i, \mathsf{req}_M)$: It executes $\mathsf{st}_{\mathsf{chal}}, \mathsf{chal}_M \leftarrow \mathsf{attestchal}_{\mathsf{ac}}(\mathsf{req}_M, \mathsf{sk}_M, \mathsf{pk}_M)$ and returns chal_M to $\mathcal{A}$ and sets $st_M[M^i] := \mathsf{st}_{\mathsf{chal}}$.
• $\mathsf{MedResp}(T, \mathsf{chal}_M)$: The oracle executes $\mathsf{resp}_M \leftarrow \mathsf{attestresp}_{\mathsf{ac}}(\mathsf{chal}_M, \mathsf{ask}_T)$ and forwards the output to $\mathcal{A}$.
• $\mathsf{MedAttest}(M^i, \mathsf{resp}_M)$: The oracle executes $\mathsf{st}_{\mathsf{chal}} := st_M[M^i]$ and $(att_m, \sigma_m) \leftarrow \mathsf{attest}_{\mathsf{ac}}(\mathsf{st}_{\mathsf{chal}}, \mathsf{resp}_M, \mathsf{sk}_M)$. The result (att_m, σ_m) is returned to $\mathcal{A}$.

4.2.2 Security and Privacy of PAwAM

We will now define the security experiment for passwordless authentication with attributes and a mediator. All definitions follow the same pattern as in the standard PAwA case, except that we provide the adversary with means to simulate the interaction between tokens and the mediator. As mentioned, we must also introduce security notions that will capture a malicious mediator that tries to break the system's privacy (e.g., learning the attributes or the origin of the server).

Definition 10 (Impersonation Security)**.** For the PAwAM, the impersonation security experiment, $\mathbf{Imp}^{\mathcal{A}}_{\mathsf{PAwAM}}$ is the same as defined for PAwA, except that the adversary $\mathcal{A}$ is given access to the oracles from Definition 9 during the online phase.

Definition 11 (Attribute Unforgeability)**.** For PAwAM, the attribute unforgeability experiment, $\mathbf{Att\text{-}Unf}^{\mathcal{A}}_{\mathsf{PAwAM}}$ is the same as defined for PAwA, except that $\mathcal{A}$ is given access to the oracles from Definition 9 during the online phase. We define as an additional winning condition that there is no query made to MedReq using challenge c for the server handle $\pi^{i,j}_S$.

Definition 12 (Unlinkability)**.** For the PAwAM, the unlinkability experiment, $\mathbf{Unl}^{\mathcal{A}}_{\mathsf{PAwAM}}$ is the same as defined for PAwA, except that the adversary $\mathcal{A}$ is given access to the oracles from Definition 9 during the phases 1 and 3.

We will now introduce two notions for PAwAM that informally capture the following two privacy concerns. First, we ensure that given an attestation request, the mediator cannot distinguish which origin the user is trying to access. Second, the mediator should attest to the user's attributes without learning anything about them. We capture the first notion informally with an experiment where the adversary specifies one token and two servers. The experiment then proceeds with picking one of the servers and starting the interaction with the adversary that plays the role of the mediator. The adversary wins if it can guess which server the user tried to access. We call this notion origin privacy and define it more formally in Definition 13. In addition to origin privacy, we define attribute privacy that captures the latter informal property. The adversary now picks two tokens and one server. We want to model that no information about the attributes of the two tokens is leaked to the mediator. Therefore, the experiment randomly picks one of the tokens and asks the adversary to attest the token to the chosen server. Here we distinguish two versions of attribute privacy: one-time and many-time. Both ensure that the attributes used are hidden from the mediator, but in one-time privacy, attestation requests from the same token are linkable. Notably, this is similar to one-time-show and multi-show security of anonymous credentials, where in the former showing attributes twice is linkable. Still, the attributes (e.g., personal data) are hidden, and there is no link to the issuing process of the credentials. Although the mediator can see that the same token/user is trying to receive an attestation, the server cannot do this. Moreover, implementing a local mediator is a simple solution to make any one-time scheme many-time secure. In Section 6.1, we provide threat analysis that depends on how the mediator is implemented in practice.

We assume that the mediator does not collude with the servers or the tokens, and further security analysis of the collusion is provided in Section 6.1.

Definition 13 (Origin Privacy)**.** For the PAwAM, the origin privacy experiment, $\mathbf{Orig\text{-}Priv}^{\mathcal{A}}_{\mathsf{PAwAM}}$ is defined to run between the challenger and an adversary $\mathcal{A}$.

- **Setup.** For each token $T \in \mathcal{T}$, a key is generated by running $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. The adversary assigns the attribute set for tokens and policy set for servers by calling the oracle Setup and passing in the attribute set lists.
- **Phase 1.** The adversary is allowed to interact with oracles Start, Complete, Challenge (see Definition 3) and oracles MedReq, MedResp.
- **Challenge Phase.** The adversary outputs a token identifier T and two (not necessarily distinct) server identifiers $S_0, S_1 \in \mathcal{S}$. The experiment chooses a bit b uniformly at random and runs $(c_b, \cdot) \leftarrow \mathsf{achall}_{\mathsf{ac}}(\mathsf{ask}_T, \mathsf{id}_{S_b})$ and $(\mathsf{nonce}_b, \mathsf{req}_{M,b}) \leftarrow \mathsf{attestreq}_{\mathsf{ac}}(\mathsf{ask}_T, c_b)$. The experiment gives $\mathsf{req}_{M,b}$ to the adversary. The adversary outputs a challenge $\mathsf{chal}_{M,b}$ to which the experiment responds with $\mathsf{resp}_{M,b}$, where $\mathsf{resp}_{M,b} \leftarrow \mathsf{attestresp}_{\mathsf{ac}}(\mathsf{ask}_T, \mathsf{chal}_{M,b})$.
- **Output Phase** Finally, the adversary outputs a bit $\hat{b}$. The experiment returns 1 if and only if bit $\hat{b}$ is equal to bit b.

Definition 14 (One-time Attribute Privacy)**.** For the PAwAM, the attribute privacy experiment, $\mathbf{Att\text{-}Priv}^{\mathcal{A}}_{\mathsf{PAwAM}}$ is defined to run between the challenger and an adversary $\mathcal{A}$.

- **Setup.** For each token $T \in \mathcal{T}$, a key is generated by running $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. The adversary assigns the attribute set for tokens and policy set for servers by calling the oracle Setup and passing in the attribute set lists.
- **Phase 1.** The adversary can interact with oracles Start, Complete, Challenge (see Definition 3). Additionally, it can interact with oracles MedReq, MedResp.
- **Challenge Phase 1.** The adversary outputs two (not necessarily distinct) token identifiers T_0, T_1, and one server identifier $S \in \mathcal{S}$.
- **Challenge Phase 2.** The experiment refreshes the attribute-based secret keys $\mathsf{ask}_{T_0}, \mathsf{ask}_{T_1}$ for tokens T_0, T_1 by running $\mathsf{ask}_{T_0} \leftarrow \mathsf{IssCred}(\mathsf{Att}_{T_0})$ and $\mathsf{ask}_{T_1} \leftarrow \mathsf{IssCred}(\mathsf{Att}_{T_1})$.
- **Challenge Phase 3.** The experiment chooses a bit b uniformly at random and runs: $(c, \cdot) \leftarrow \mathsf{achall}_{\mathsf{ac}}(\mathsf{id}_S)$ and $(\mathsf{nonce}_b, \mathsf{req}_{M,b}) \leftarrow \mathsf{attestreq}_{\mathsf{ac}}(\mathsf{ask}_{T_b}, c)$. The experiment gives $\mathsf{req}_{M,b}$ to the adversary. The adversary outputs a challenge $\mathsf{chal}_{M,b}$ to which the experiment responds with $\mathsf{resp}_{M,b}$, where $\mathsf{resp}_{M,b} \leftarrow \mathsf{attestresp}_{\mathsf{ac}}(\mathsf{chal}_{M,b}, \mathsf{ask}_{T_b})$.
- **Output Phase** Finally, the adversary outputs a bit $\hat{b}$. The experiment returns 1 if and only if bit $\hat{b}$ is equal to bit b.

Definition 15 (Many-times Attribute Privacy)**.** We define many-times attribute privacy similar to one-time attribute privacy, except that it omits **Challenge Phase 2**.

5 FIDO-AC: System Design

In this section, we will describe the FIDO-AC system. First, we give an overview of actors and interactions. We then describe the requirements that an anonymous credential system must support to be used in FIDO-AC. We show this in the example of an electronic identity document (eID) in the ICAO

```
$\textbf{attestreq}_{\textbf{ac}}(\mathsf{ask}_T, c)$
    $\mathsf{nonce} \xleftarrow{\$} \{0,1\}^{\lambda}$
    $(\mathsf{H}(DG), pk_{eID}, \pi_{PA}) \leftarrow BAC/PACE(\mathsf{ask}_T)$
    $\mathsf{req}_M := (\mathsf{H}(DG), \mathsf{pk}_{eID}, \pi_{PA}, c, \mathsf{nonce})$
    $ret\ \mathsf{nonce}, \mathsf{req}_M$
```

```
$\textbf{attestchal}_{\textbf{ac}}(\mathsf{req}_M, \mathsf{sk}_M, \mathsf{pk}_M)$
    $(\mathsf{H}(DG), \mathsf{pk}_{eID}, \pi_{PA}, c, \mathsf{nonce}) := \mathsf{req}_M$
    $key_{ses} \leftarrow KE(\mathsf{pk}_{eID}, \mathsf{sk}_M)$
    $cmd_{cha} \leftarrow AE\text{-}ENC(key_{ses}, cmd)$
    $\mathsf{chal}_M := (\mathsf{pk}_M, cmd_{cha})$
    $\mathsf{st}_{\mathsf{chal}} := (\mathsf{req}_M, key_{ses})$
    $ret\ \mathsf{st}_{\mathsf{chal}}, \mathsf{chal}_M$
```

```
$\textbf{attestresp}_{\textbf{ac}}(\mathsf{chal}_M, \mathsf{ask}_T)$
    $(\mathsf{pk}_M, cmd_{cha}) := \mathsf{chal}_M$
    $\mathsf{resp}_M \leftarrow CA(\mathsf{pk}_M, cmd_{cha}, \mathsf{ask}_T)$
    $ret\ \mathsf{resp}_M$
```

```
$\textbf{attest}_{\textbf{ac}}(\mathsf{st}_{\mathsf{chal}}, \mathsf{resp}_M, \mathsf{sk}_M)$
    $(\mathsf{req}_M, key_{ses}) := \mathsf{st}_{\mathsf{chal}}$
    $(\mathsf{H}(DG), \mathsf{pk}_{eID}, \pi_{PA}, c, \mathsf{nonce}) := \mathsf{req}_M$
    $b_{PA} \leftarrow PA_{verify}(\mathsf{H}(DG), \mathsf{pk}_{eID}, \pi_{PA})$
    $b_{CA} \leftarrow CA_{Verify}(\mathsf{resp}_M, key_{ses})$
    $att_m := \mathsf{H}(\mathsf{H}(DG)||\mathsf{nonce})||c$
    $\sigma_m := \perp$
    $\sigma_m \leftarrow \mathsf{Sign}(\mathsf{sk}_M, att_m)\ if\ (b_{PA} \wedge b_{CA})$
    $ret\ att_m, \sigma_m$
```

```
$\textbf{prove}_{\textbf{ac}}(att_m, \sigma_m, \mathsf{nonce}, \mathsf{Att}, \mathsf{Policy}_S)$
    $DG \leftarrow Parse(\mathsf{Att})$
    $(m||c_m) := att_m$
    $\pi_{\mathsf{zkp}} \leftarrow \mathsf{ZKProve}(\mathsf{crs}, (m, \mathsf{Policy}_S),$
              $(DG, \mathsf{nonce}))$
    $\Pi_{\mathsf{Att}} := (att_m, \sigma_m, \pi_{\mathsf{zkp}})$
    $ret\ \Pi_{\mathsf{Att}}$
```

```
$\textbf{check}_{\textbf{ac}}(\Pi_{\mathsf{Att}}, \mathsf{Policy}_S, \mathsf{pk}_M, c)$
    $(att_m, \sigma_m, \pi_{\mathsf{zkp}}) := \Pi_{\mathsf{Att}}$
    $b_M \leftarrow \mathsf{Verify}(\mathsf{pk}_M, att_m, \sigma_m)$
    $(m||c_m) := att_m$
    $b_{zkp} \leftarrow \mathsf{ZKVer}(\mathsf{crs}, (m, \mathsf{Policy}_S))$
    $b_{challenge} \leftarrow c_m \overset{?}{=} c$
    $b_{ac} \leftarrow b_M \wedge b_{zkp} \wedge b_{challenge}$
    $ret\ b_{ac}$
```

Table 1: The pseudocode of FIDO-AC for the algorithms defined in Definition 8. Wrapper algorithms: KE - key exchange, $AE-ENC$ - authenticated encryption, PA_{verify} - passive authentication verification, CA_{verify} - chip authentication verification.

standard. Finally, we show how those credentials can be integrated into the FIDO2 authentication process.

5.1 Overview

FIDO-AC is a novel system that satisfies the requirements and threat model stated in Section 3. Figure 3 illustrates the high-level overview of the new modules introduced by the FIDO-AC extension to the existing FIDO2. At its core, the FIDO-AC system comprises three subsystems, namely anonymous credentials (AC), mediator, and FIDO extension integrated to provide the needed functionalities. The AC module will gather and supply all the necessary information required to fully realize an AC system while utilizing the binding to FIDO as the medium to deliver the information. Compared to using the standard FIDO2 protocol, users have to install an additional FIDO-AC application and scan their eID for proof of interaction. The application is also used to arbitrarily disclose the attributes of users, using a non-interactive zero-knowledge proof system. Note that installation is a one-time setup, while the result from the scanning of eID can be reused if the user explicitly permits caching. Additionally, the FIDO-AC framework relies on a trusted mediator to produce the proof of interaction with an eID for each FIDO2 transaction.

Below we describe the FIDO-AC protocol (illustrated in Figure 2) using the notion of passwordless authentication with attributes and mediator we introduced in the previous section. The description will use the standard FIDO passwordless authentication as a building block. Thus, we will use the notion provided by Hanzlik et al. [21]: rchall, achall, rcomm, acomm, rresp, aresp, rcheck, acheck, to denote the standard protocol and use the suffix *ac* (e.g., $\mathsf{rchall}_{\mathsf{ac}}$) to indicate our PAwAM. Please note that only $\mathsf{rchall}_{\mathsf{ac}}$, $\mathsf{achall}_{\mathsf{ac}}$, $\mathsf{rcomm}_{\mathsf{ac}}$, $\mathsf{acomm}_{\mathsf{ac}}$, and $\mathsf{rcheck}_{\mathsf{ac}}$, $\mathsf{acheck}_{\mathsf{ac}}$ introduce additional steps to the FIDO protocol, which remain the same for registration and authentication processes. Therefore, we only provide descriptions of the functions mentioned above and the pseudocode for the new algorithms (see Table 1).

The $\mathsf{rcomm}_{\mathsf{ac}}$ and $\mathsf{acomm}_{\mathsf{ac}}$ algorithms are triggered with server identity id_S and challenge c_p (the standard FIDO challenge is extended with a policy). Then, the client extracts data from the eID (see step † in Figure 2 and Section 5.2), which is followed by a communication with the mediator to run the liveliness check (steps ‡) and an attestation generation (step *). Then, the client runs the zero-knowledge proof (ZKP) generation (step **). The attestation is hashed and appended to the challenge, which is passed to the rcomm or acomm functions. Regarding the $\mathsf{rcheck}_{\mathsf{ac}}$ and $\mathsf{acheck}_{\mathsf{ac}}$ functions, we modify the generated challenge with the hashed attestation values and ZKP. Then, we use the modified challenge in the rcheck and acheck algorithms. Finally, we run a $\mathsf{check}_{\mathsf{ac}}$ algorithm, which verifies both ZKP and attestation. The server's decision depends on both FIDO and FIDO-AC checks.

5.2 Anonymous Credentials

FIDO-AC can use any anonymous credential system supporting an active non-shareability test. In literature, this is usually ensured via binding the secret key for the credentials to a hardware token. Due to the construction of the FIDO-AC framework, it can not only support any attribute system with a non-shareability property but also improve the privacy guarantees of the final solution. Without loss of generality, we will use ICAO-compliant eIDs as the basis for the credential part of FIDO-AC. The way we will use the eID can be described as follows. A FIDO-AC-specific application will extract the authenticated data for the eID. A mediator can verify the data using the Passive Authentication protocol described in Section 2.2, and then run the liveliness check.

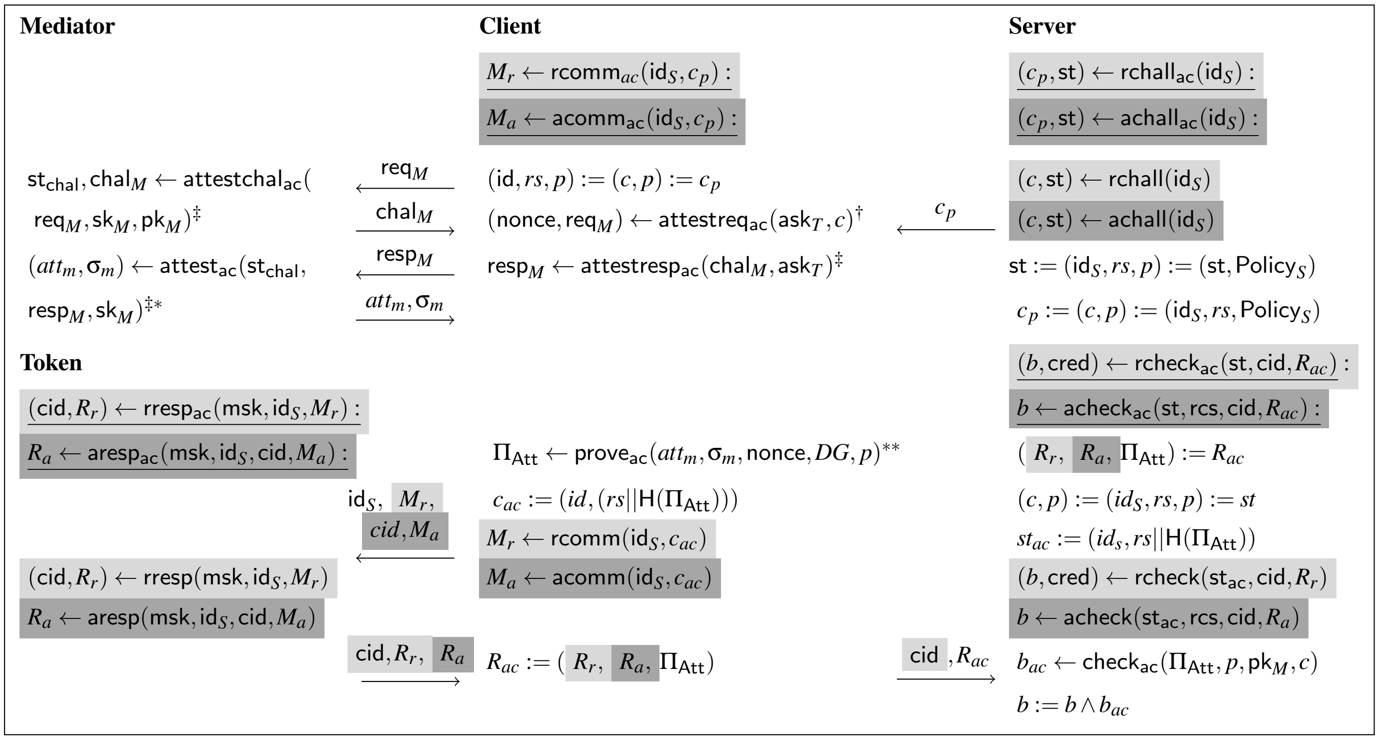
Mediator
Client
Server
$M_r \leftarrow \mathsf{rcomm}_{ac}(\mathsf{id}_S, c_p)$:
$M_a \leftarrow \mathsf{acomm}_{ac}(\mathsf{id}_S, c_p)$:
$(c_p, \mathsf{st}) \leftarrow \mathsf{rchall}_{ac}(\mathsf{id}_S)$:
$(c_p, \mathsf{st}) \leftarrow \mathsf{achall}_{ac}(\mathsf{id}_S)$:
$\mathsf{st}_{chal}, \mathsf{chal}_M \leftarrow \mathsf{attestchal}_{ac}(\mathsf{req}_M, \mathsf{sk}_M, \mathsf{pk}_M)^{\ddagger}$
req_M
chal_M
resp_M
att_m, σ_m
$(\mathsf{id}, rs, p) := (c, p) := c_p$
$(\mathsf{nonce}, \mathsf{req}_M) \leftarrow \mathsf{attestreq}_{ac}(\mathsf{ask}_T, c)^{\dagger}$
c_p
$(c, \mathsf{st}) \leftarrow \mathsf{rchall}(\mathsf{id}_S)$
$(c, \mathsf{st}) \leftarrow \mathsf{achall}(\mathsf{id}_S)$
$(att_m, \sigma_m) \leftarrow \mathsf{attest}_{ac}(\mathsf{st}_{chal}, \mathsf{resp}_M, \mathsf{sk}_M)^{\ddagger *}$
$\mathsf{resp}_M \leftarrow \mathsf{attestresp}_{ac}(\mathsf{chal}_M, \mathsf{ask}_T)^{\ddagger}$
$\mathsf{st} := (\mathsf{id}_S, rs, p) := (\mathsf{st}, \mathsf{Policy}_S)$
$c_p := (c, p) := (\mathsf{id}_S, rs, \mathsf{Policy}_S)$
Token
$(b, \mathsf{cred}) \leftarrow \mathsf{rcheck}_{ac}(\mathsf{st}, \mathsf{cid}, R_{ac})$:
$b \leftarrow \mathsf{acheck}_{ac}(\mathsf{st}, \mathsf{rcs}, \mathsf{cid}, R_{ac})$:
$(\mathsf{cid}, R_r) \leftarrow \mathsf{rresp}_{ac}(\mathsf{msk}, \mathsf{id}_S, M_r)$:
$R_a \leftarrow \mathsf{aresp}_{ac}(\mathsf{msk}, \mathsf{id}_S, \mathsf{cid}, M_a)$:
$\Pi_{\mathsf{Att}} \leftarrow \mathsf{prove}_{ac}(att_m, \sigma_m, \mathsf{nonce}, DG, p)^{**}$
$(R_r, R_a, \Pi_{\mathsf{Att}}) := R_{ac}$
$\mathsf{id}_S, M_r, cid, M_a$
$c_{ac} := (id, (rs || \mathsf{H}(\Pi_{\mathsf{Att}})))$
$(c, p) := (id_S, rs, p) := st$
$M_r \leftarrow \mathsf{rcomm}(\mathsf{id}_S, c_{ac})$
$st_{ac} := (id_s, rs || \mathsf{H}(\Pi_{\mathsf{Att}}))$
$(\mathsf{cid}, R_r) \leftarrow \mathsf{rresp}(\mathsf{msk}, \mathsf{id}_S, M_r)$
$M_a \leftarrow \mathsf{acomm}(\mathsf{id}_S, c_{ac})$
$(b, \mathsf{cred}) \leftarrow \mathsf{rcheck}(\mathsf{st}_{ac}, \mathsf{cid}, R_r)$
$R_a \leftarrow \mathsf{aresp}(\mathsf{msk}, \mathsf{id}_S, \mathsf{cid}, M_a)$
$b \leftarrow \mathsf{acheck}(\mathsf{st}_{ac}, \mathsf{rcs}, \mathsf{cid}, R_a)$
cid, R_r, R_a
$R_{ac} := (R_r, R_a, \Pi_{\mathsf{Att}})$
cid, R_{ac}
$b_{ac} \leftarrow \mathsf{check}_{ac}(\Pi_{\mathsf{Att}}, p, \mathsf{pk}_M, c)$
$b := b \wedge b_{ac}$

Figure 2: The FIDO-AC protocol for registration and authentication. The flow reuses definitions in [21] (Figure 1). ▭ - registration, ▨ - authentication, if not marked, applicable to both flows. † - eID read, ‡ - liveliness check, * - mediator attestation, ** - ZKP generation.

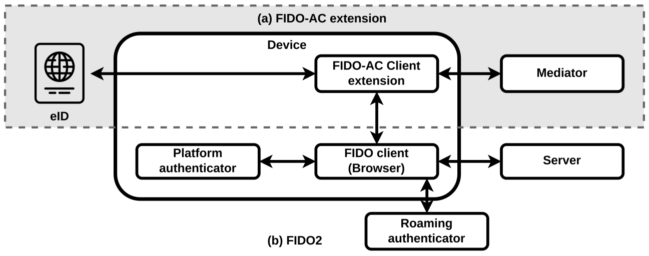

Figure 3: Differences between (a) FIDO-AC and (b) FIDO2. Additional parts of FIDO-AC are given in the gray box.

The liveliness check, used by mediator, verifies whether the user owns the provided authenticated data. Such a check is implemented in the ICAO-based eID infrastructure using Active Authentication or its variant that we will use namely Chip Authentication (CA). The idea behind those authentication methods is that eID is equipped with a secret key stored in secure memory. In the eID setting, it is assumed that the key never leaves this secure memory and cannot be extracted. During CA, the eID proves knowledge of the secret key with respect to a public key bound to the authenticated data.

We will now summarize the liveliness test in more detail. The test starts with the eID sending the hash value of the data it stores, including the public key and the issuer's signature (see Section 2.2 for more information), as well as the relying party's challenge and application's nonce for binding purposes. Using this data, the mediator performs both the CA and PA. After the session keys are replaced, the mediator queries the eID for a random challenge for the Terminal Authentication (TA) protocol. This command is encrypted using the session keys that are the result of the CA protocol. Thus, it implicitly prove knowledge of the secret key corresponding to the disclosed eID public key, which is bound to authenticated data. The result of the liveliness test is a signature of the mediator attesting that it performed the liveliness test for the signed data. The liveliness test is captured in the algorithms ($\mathsf{attestreq}_{ac}$,$\mathsf{attestchal}_{ac}$,$\mathsf{attestresp}_{ac}$,attest_{ac}), and the pseudocode for the algorithms can be found in Table 1.

Disclosing Attributes, in the FIDO-AC system, is implemented using zero-knowledge (ZK) proofs. Recall that the mediator's signature is under a specific hash value and the relying party's challenge. This hash value is a salted hash of the hashed attributes of the user data and, as such, contains data that should not be disclosed to the verifier. At the same time, the verifier must check this double-hashed salted data to adhere to some policies. Therefore, we will use a proof system to do exactly this.

To limit communication, we consider using the non-interactive zero-knowledge proof (NIZK) system, as it do not require interaction between the prover and the verifier. There exist many non-interactive ZK-proof systems supporting arbitrary computation to be proven, namely Groth'16 ZK-SNARK [19], the setup less bulletproof [7], and ZK-STARK [5]. The exact choice of the proof system is not pertinent to the design and is left as the implementation details. The anonymous credential in FIDO-AC will take the form of a non-interactive zero-knowledge proof about some properties of the data from the eID, which are represented by a double-hashed salted value. The credential will also include the attestation of the mediator in the form of a standard digital signature on this hash value that will be returned as the result of the liveliness test. The ZK proof of attributes is captured in the algorithms ($\mathsf{prove}_{\mathsf{ac}}$,$\mathsf{check}_{\mathsf{ac}}$), and the pseudocode for the algorithms can be found in Table 1.

5.3 FIDO-AC extension

FIDO2 extensions are the recommended way to extend FIDO2 functionality. Therefore our design follows the best practices and introduces a new extension called *fidoac*. However, including a new extension to the FIDO2 messages is not enough to fully integrate with the FIDO2 ecosystem. Our objective is to minimize the effort of adapting the FIDO-AC system and provide a smooth integration with existing FIDO clients and authenticators. Therefore, we analyzed the available solutions to select the most suitable approach for the FIDO-AC system. We identified three approaches: custom client [20], relying party modification [27], and client extension [29] (detailed evaluation in Appendix D). However, none of the above-mentioned solutions is sufficient to design an architecture that fulfills our requirements. Therefore, we followed a hybrid approach based on the modifications introduced before and after WebAuthn API is called, and marginal modification in the FIDO server. First, the FIDO assertion request needs to be extended to include *fidoac* extension. Second, an additional JavaScript (*fidoac.js*) needs to be included in the page, which handles the *navigator.credentials.get* execution. Finally, the FIDO server needs an additional code snippet used to verify the FIDO assertion with the *fidoac* extension. FIDO-AC modified the challenge used, and a detailed discussion of the modification can be found in Appendix C.

Our approach is fully compatible with FIDO2 (**R.3**) and does not modify user-facing FIDO2 parties (**R.5**). Thus, it can be considered to be as scalable as a pure FIDO2 system (**R.6**). The modifications of the FIDO server are trivial to implement with the existing FIDO2 libraries (e.g., custom extensions in SimpleWebAuthn Server[5]). Similarly, the verification modification requires only two widely supported primitives (SHA-256 and base64 encoding). Regarding *fidoac.js*, the JavaScript can be imported directly from an external source (e.g., hosted by FIDO-AC) to the web page. Therefore, we claim that the introduced modifications fulfil requirement **R.5**.

6 Security Analysis

In this section, we analyze the security and privacy of FIDO-AC and only sketch the idea behind why security holds and give complete proof in the full paper.

[5]https://www.npmjs.com/package/@simplewebauthn/server

Lemma 1. The FIDO-AC protocol presented in Figure 2 is secure against impersonation as defined in Definition 10 assuming the underlying passwordless authentication PLA protocol used as a building block is secure against impersonation.

Proof sketch. The idea is that the FIDO protocol remains unchanged in FIDO-AC, which only adds additional parts that do not influence impersonation resistance.

Lemma 2. The FIDO-AC protocol presented in Figure 2 is unlinkable as defined in Definition 12 assuming the underlying passwordless authentication PLA protocol is secure against unlinkability, the hash function H is a random oracle and the used proof system is zero-knowledge.

Proof sketch. The only additional data compared to a standard FIDO response is the mediator's attestation and zero-knowledge proof. The former is a standard signature on the message $att_m = (\mathsf{H}(\mathsf{H}(DG)||nonce)||c)$, which leaks no information about the user's attributes stored in DG due to the use of a random *nonce*. Finally, the proof provided is zero-knowledge, also leaking no additional information.

Lemma 3. The FIDO-AC protocol presented in Figure 2 is attribute unforgeable as defined in Definition 11 assuming the unforgeability of the mediator signature, the security the eID and the soundness of the used proof system.

Proof sketch. For the server to accept in $\mathsf{check}_{\mathsf{ac}}$, the adversary must be able to provide zero-knowledge proof proving the message signed by the mediator contains attributes satisfying the server's policy. Since the proof system is sound, the only way to win is for the adversary to forge a signature on the mediator's behalf or to break the security of the eID (i.e., create a fake eID that passes as valid).

Lemma 4. The FIDO-AC protocol presented in Figure 2 satisfies origin privacy as defined in Definition 13.

Proof sketch. The mediator only gets access to the server's challenge and information about the token, which is independent of the server in this experiment. The honest server chooses the challenge uniformly at random, so it does not reveal any information about it.

Lemma 5. The FIDO-AC protocol presented in Figure 2 satisfies one-time attribute privacy as defined in Definition 14 assuming the random oracle model and the unlinkability of data groups for the same personal data of the eID.

Proof sketch. The mediator learns the hash value of the data groups and a static public key for one of the tokens. Since we only have to prove one-time security, the data groups of both challenged tokens are reissued, and an adversary can break attribute privacy only if it can distinguish the refreshed data groups of the tokens (i.e., break the assumption for eID).

6.1 Mediator Threat Analysis

The Mediator in FIDO-AC can be instantiated in multiple ways. We first rule out the possibility of delegating the medi-

	Mediator-Verifier	Mediator-Prover
Unlinkability	None: ✗* TEE: ✗* C-TEE: ✓	✓
Attribute Unforgeability	✓	None: ✗ TEE: ✓ C-TEE: ✓

Table 2: Unlinkability and attribute unforgeability properties for colluding parties of the FIDO-AC system.
* - considering ICAO eID, for other eID schemes a stronger unlinkability property can be achieved

ator role to the client's browser, as it lacks a trusted execution environment (i.e., the verifier cannot trust such a mediator), and the configuration of the relying party acting as a mediator, as it breaks the privacy assumptions (i.e., linking user using identifiable properties of eID). Therefore, we only consider the following versions of mediator configuration: a local application (backed with hardware attestation), a remote trusted third party, or any party that uses confidential TEE.

The mediator (local or remote) colluding with either verifier or prover introduces a threat of breaking the properties of the FIDO-AC system. In Table 2, we evaluate privacy (i.e., unlinkability) and attribute unforgeability properties considering colluding mediator and various execution environments. The collaboration of the mediator and verifier breaks the unlinkability property because data received by the verifier can be linked to the corresponding eID, unless a confidential TEE is used (i.e., user identifier not revealed to the mediator). For the mediator colluding with the prover, the attribute unforgeability property can be compromised if the TEE environment is not used (i.e., any proof can be generated). Therefore, we claim that a local verifier provides better privacy guarantees for privacy-oriented systems, whereas a remote mediator is more suitable for highly secure configurations.

The practical implementation of mediators relies on a secure trusted environment. Notably, we acknowledge the threats of jailbreaking TEE or leaking information from confidential TEE (e.g., side-channel attacks for SGX enclave [10, 30]). However, we argue that for the majority of use cases, we can safely assume that the TEE properties hold. Regarding trusted third party, we argue that, though it guarantees privacy and soundness of the system, it does not provide incentives for the running entity, and thus reduces the practical value. Therefore, considering the above threats, we decided to implement the FIDO-AC system (see Section 7) with a local mediator.

6.2 Web Security

The FIDO-AC system alters the execution of the application on the client side (i.e., web browser) by introducing a *fidoac.js* library. The security of loading and executing the *fidoac.js* script, similarly to any JavaScript library, depends on the deployment method (e.g., CDN or same origin) and applied web security mechanisms such as Subresource Integrity (i.e., verifying the script's hash) and Content Security Policy (i.e., restricting the script's origin). Considering the integration method of *fidoac.js* (i.e., a decorator pattern), we argue that our modification of the *navigator.credentials* object does not change the security properties of the parent application. Additionally, we consider the security of the communication between *fidoac.js* and the FIDO-AC client extension (e.g., FIDO-AC mobile application) depends on the platform-specific mechanisms, however, unintentional verification is unlikely because sharing the credentials involves user actions (e.g., providing an eID to verify liveliness). Nevertheless, if the channel is insecure, an adversary could send the request at the right time (i.e., during a valid transaction) to launch a hijacking attack, hoping that the user does not notice any difference in the transaction data (similarly to MFA fatigue attacks). Therefore, we claim that the security of *fidoac.js* extension relies only on the web application configuration, browser security, and OS means to communicate with the FIDO-AC client extension.

7 Implementation and Evaluation

In this section, we describe our approach to building the FIDO-AC system and demonstrate its feasibility in practical deployment in terms of performance evaluation. Notably, our prototype is one possible instantiation, and we can effortlessly adjust to any requirements. Following the system requirements (see Section 3), our implementation is suitable for most existing FIDO2 deployments. We achieved the claimed requirements by encapsulating and extracting the FIDO-AC-specific logic and then introducing the integration points. The high-level view of our FIDO-AC system implementation is presented in Figure 4. More details on the interaction between different components can be found in Appendix A. A detailed description of the system elements and the discussion about the design decisions can be found in Appendix B. The source code is published in our open-sourced repository (`https://github.com/FIDO-AC/fidoac`).

7.1 Implementation

The FIDO-AC system implementation is based on three components. The first is a user-centered mobile application for Android OS that introduces a bridge between the eID (in our prototype, an ICAO-based ePassport) and the FIDO authentication mechanism. We reuse an NFC interface to execute the PACE (or alternatively BAC) protocol and implement usability enchancements such as extracting data from the machine readable zone (MRZ) and data caching. We decided to follow a local mediator approach which is designed as a part of the FIDO-AC application. It relies on the security assumptions of Android's TEE. The zero-knowledge proof generation functionality follows the Groth'16 ZK-SNARK [19] using ported

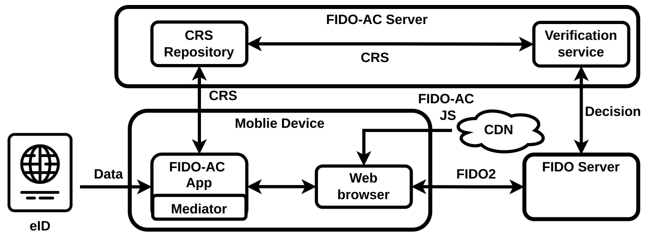

Figure 4: FIDO-AC system implementation high-level view

versions of rust-arkwork [12] and libsnark [26] libraries. The security as well as ZKP selection considerations can be found in Appendix B). The second component, FIDO-AC Server, is introduced to simplify the integration with existing FIDO implementations. It facilitates the anonymous credential trusted setup and verification. We recognized that the centralized and externalized party for the server-side process (e.g., ZKP verification), contributes to the usability and deployability of the FIDO-AC system. Similarly, the JavaScript integration (i.e., *fidoac.js*) is prepared to seamlessly introduce the FIDO-AC modifications to the browser execution. The script is served from CDN and automatically overrides the *navigator.credential* functions.

7.2 Performance Evaluation

We tested the FIDO-AC system using a Google Pixel 6 Pro and a Standard D4s v3 Microsoft Azure cloud instance (4 vcpus, 16 GiB memory). Without loss of generality, we use an ePassport as the ICAO-compliance eID. The performance of the running time is across 100 runs of the measured component. Reading the eID data takes about $\sim 1050ms$ in itself. Fortunately, this operation can be cached in the application's memory. The other part of the interaction with the eID (i.e., secure channel establishment via BAC/PACE and liveliness test with a local mediator) takes $\sim 740ms$. Verifying the ZKP is the least taxing operation, which only takes $< 10ms$ due to the usage of ZK-SNARK. The downside of using ZK-SNARKs is the proving time. The proving time for our example (i.e., data and age policy proof) is $\sim 3.3s$ which is relatively slow compared to the verification time. Fortunately, the FIDO-AC application can precompute such proofs since they do not depend on any values chosen by third parties but only on data and randomness chosen by the application. The application can do it since the space of practical queries is small, or the user can predefine a set of accepted predicates. Assuming a completed offline preprocessed proof and cached eID data, the added latency of the FIDO-AC system compared to standard FIDO2 is less than $1s$ for our implementation.

8 Discussion

Achieving high usability was the primary goal of our design. Therefore, we inspect the usability of FIDO-AC compared to other schemes. The user must install a separate FIDO-AC application to use the FIDO-AC system. Note that different relying parties' services can reuse the same application. Thus, the initial setup phase is one-time for all future FIDO-AC-enabled services. To reduce user friction and improve usability, users can opt into an optional document data caching feature to prevent inputting the document data in subsequent runs if they are comfortable with it. The FIDO-AC application can provide optical character recognition (OCR) functionality to read the necessary information from the document's machine-readable zone (MRZ), eliminating the need for error-prone and high-friction manual input. It is worth mentioning that the usability of FIDO-AC (with cached eID data) is comparable to the FIDO2 authentication via the NFC channel (i.e., eID must be placed close to the reader).

Operation	Platform	Time (ms)	SD (ms)
eID Reading	Mobile	1059.4/0.0†	37.58
Liveliness Check	Mobile	738.92	47.06
ZK Verify	Cloud PC	8.19	0.29
ZK Prove	Mobile	3375.61*	95.25

† The FIDO-AC application can cache the read data.
* Running time reduces significantly with preprocessing.

Table 3: Performance overview of the various FIDO-AC operations. Time is averaged over 100 executions.

In the case of the relying parties, we focused our efforts on the usability of deployment and integration. The relying party only has to include the provided JavaScript file into the web application page containing the call to the browser WebAuthn API without making any other changes to the existing web application codebase. To simplify and smoothen the integration of the verification logic needed by FIDO-AC into the existing FIDO infrastructure, we provide a docker image containing the verification service and its dependencies are provided. Alternatively, the relying party can choose to invoke the verification service hosted by the FIDO-AC server and parse the result according to the relevant business logic, which further minimizes the integration effort.

Electronic Identification Schemes. Digital transformation has been a strategic target for many countries, including issuing digital identity documents and remote identity verification. In particular, frameworks following the Issuer, Verifier, and Holder model, such as mobile driving license (mDL) [23] or eIDAS EUDI Wallet [11], are being introduced as a legal means to identify people. The FIDO-AC framework (excluding privacy advancements) resembles the abovementioned schemes with some key differences. FIDO-AC leverages existing eID documents, and thus it does not mandate to have a Holder role. In consequence, FIDO-AC does not introduce additional provisioning and enrollment procedures. Similarly, unlike generic eID frameworks, FIDO-AC proposes a specific set of technologies (e.g., ZKP and FIDO2) that, even though might not be suitable for every deployment, reduces the development and interoperability efforts. Using external eID, FIDO-AC implements roaming attributes (a concept similar to the FIDO2 roaming authenticator), enabling device-independent identity verification (e.g., through thin clients or kiosks). Regarding privacy, the mediator-based setup of FIDO-AC allows for a multi-show of credentials, which cannot be easily achieved with static credentials (e.g., a new set of credentials has to be issued to remain private, which is a noticeable inconvenience for both Issuer and Holder). Notably, FIDO-AC and wallet-based schemes such as mDL are not exclusive. On the contrary, FIDO-AC can utilize those schemes if they provide an active authentication feature.

9 Conclusion

This paper described and demonstrated an end-to-end solution for enforcing privacy in attribute-based authentication using the FIDO2 protocol. Our design considers usability for both users and implementers and is thus ready for production deployment. We integrate the eID environment and enforce a liveliness verification to increase the trustworthiness of the presented attributes, preventing the problem of attribute sharing. We leverage zero-knowledge proofs to guarantee the privacy of the attributes presentation. We introduce a custom FIDO2 extension for transporting anonymized credentials and present a mechanism to overcome the WebAuthn API limitations. We support our design with security and performance evaluations and a prototype implementation of the system components. The methods presented in this paper will contribute to sensitive data storage minimization and thus mitigate private data leaks in the future.

Acknowledgments

We thank the anonymous reviewers for Usenix Security'23 for their helpful comments and feedback. This work is supported by the Federal Ministry of Education and Research (BMBF), the German Academic Exchange Service (DAAD), and Macquarie University under the Australia–Germany Joint Research Co-operation Scheme with project ID: 57654883.

References

[1] FIDO Alliance. FIDO security reference. https://fidoalliance.org/specs/common-specs/fido-security-ref-v2.1-rd-20210525.html, 2021. [Online; accessed 11-Oct-2022].

[2] AUSTRAC. Optus Data Breach. https://www.austrac.gov.au/optus-data-breach-working-our-reporting-entities, 2022. [Online; accessed 11-Oct-2022].

[3] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. Provable security analysis of FIDO2. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 125–156, Virtual Event, August 2021. Springer, Heidelberg.

[4] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 108–125. Springer, Heidelberg, August 2009.

[5] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[6] Johannes Blömer and Jan Bobolz. Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 221–239. Springer, Heidelberg, July 2018.

[7] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[8] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 1205–1216. ACM Press, November 2014.

[9] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

[10] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. SgxPectre: Stealing intel secrets from SGX enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157, Stockholm, Sweden, June 2019. IEEE.

[11] European Commission. European digital identity architecture and reference framework – outline. Standard, European Commission, 2022.

[12] Arkworks Contributors. Arkworks zkSNARK ecosystem. https://arkworks.rs. [Online; accessed 11-Oct-2022].

[13] Geoffroy Couteau and Michael Reichle. Non-interactive keyed-verification anonymous credentials. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 66–96. Springer, Heidelberg, April 2019.

[14] Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 535–555. Springer, Heidelberg, March 2019.

[15] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018(3):164–180, July 2018.

[16] FIDO Alliance. Client to authenticator protocol (CTAP). `https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-errata-20220621.pdf`, 2021. [Online; accessed 11-Oct-2022].

[17] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In *NDSS 2014*. The Internet Society, February 2014.

[18] Google. Play Integrity API. `https://developer.android.com/google/play/integrity/overview`. [Online; accessed 11-Oct-2022].

[19] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[20] Chengqian Guo, Quanwei Cai, Qiongxiao Wang, and Jingqiang Lin. Extending registration and authentication processes of FIDO2 external authenticator with qr codes. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 518–529, 2020.

[21] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Token meets wallet: Formalizing privacy and revocation for FIDO2. In *2023 IEEE Symposium on Security and Privacy (IEEE SP)*, pages 978–995, Los Alamitos, CA, USA, May 2023. IEEE Computer Society.

[22] Scott Hendrickson, Jana Iyengar, Tommy Pauly, Steven Valdez, and Christopher A. Wood. Rate-limited token issuance protocol. Internet-Draft draft-privacypass-rate-limit-tokens-03, IETF Secretariat, July 2022.

[23] ISO Central Secretary. Personal identification — ISO-compliant driving licence — Part 5: Mobile driving licence (mDL) application. Standard ISO/IEC 18013-5:2021, International Organization for Standardization, Geneva, CH, 2021.

[24] Stephan Krenn, Thomas Lorünser, Anja Salzer, and Christoph Striecks. Towards attribute-based credentials in the cloud. In Srdjan Capkun and Sherman S. M. Chow, editors, *CANS 17*, volume 11261 of *LNCS*, pages 179–202. Springer, Heidelberg, November / December 2017.

[25] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar R. Weippl. "I have no idea what I'm doing" - on the usability of deploying HTTPS. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 1339–1356. USENIX Association, August 2017.

[26] SCIPR Lab. C++ library for zkSNARKs. `https://github.com/scipr-lab/libsnark`. [Online; accessed 11-Oct-2022].

[27] Yuto Okawa, Shuji Yamaguchi, Hidehito Gomi, and Tetsutaro Uehara. Implementation of an extended FIDO2 authenticator using attribute-based signatures. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 825–832, 2021.

[28] European Parliament. Regulation (EU) 2019/1157 of the European Parliament and of the Council of 20 june 2019 on strengthening the security of identity cards of union citizens and of residence documents issued to union citizens and their family members exercising their right of free movement (text with EEA relevance.), Jul 2019.

[29] Florentin Putz, Steffen Schön, and Matthias Hollick. Future-proof web authentication: Bring your own FIDO2 extensions. In Andrea Saracino and Paolo Mori, editors, *Emerging Technologies for Authorization and Authentication*, pages 17–32, Cham, 2021. Springer International Publishing.

[30] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. CrossTalk: Speculative data leaks across cores are real. In *2021 IEEE Symposium on Security and Privacy*, pages 1852–1867. IEEE Computer Society Press, May 2021.

[31] Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. `zk-creds`: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure. Cryptology ePrint Archive, Report 2022/878, 2022. `https://eprint.iacr.org/2022/878`.

[32] Fabian Schwarz, Khue Do, Gunnar Heide, Lucjan Hanzlik, and Christian Rossow. Feido: Recoverable FIDO2 tokens using electronic ids. In *29th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, November 2022.

[33] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *NDSS 2019*. The Internet Society, February 2019.

[34] Thales. The electronic passport in 2021 and beyond. https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/passport/electronic-passport-trends, 2021. [Online; accessed 11-Oct-2022].

[35] W3C. Web authentication: An API for accessing public key credentials level 2. https://www.w3.org/TR/webauthn-2/, 2021. [Online; accessed 11-Oct-2022].

A FIDO-AC Implementation Interactions

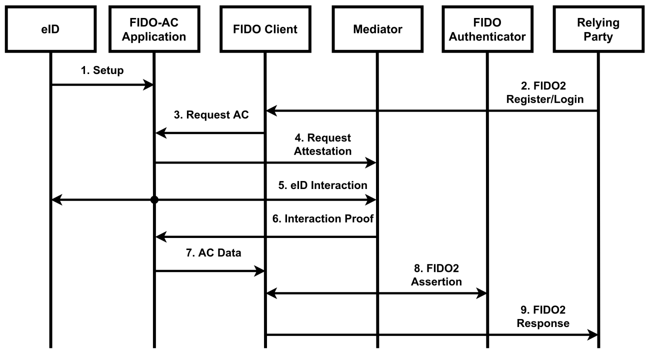

Figure 5: FIDO-AC interaction flow

Below, we describe the high-level interactions between the parties of the FIDO-AC system implementation. The illustration of the flow is presented in Figure 5. The interaction with the FIDO-AC system starts when the user bootstraps the FIDO-AC application (step 1.). The process involves reading the data and signature from the eID document and caching them in the FIDO-AC application. After the initialization, the user can trigger the FIDO2 transaction (step 2.). The FIDO2 assertion request is intercepted in the FIDO client and passed to the FIDO-AC application (step 3.). At this stage, the FIDO-AC application starts interacting with the mediator (step 4.). The application shares the eID data: the hashed data values, eID public key, and signature, as well as the FIDO challenge and nonce. Notably, the mediator does not learn the identity of the user. However, because of the eID public key, the interaction of the same eID with the mediator is linkable. The mediator's task is to verify the authenticity of the data and prove the liveliness of the eID. To accomplish that, the mediator starts the interaction with the eID (step 5.). The details of the interaction are described in Section 5.2. The mediator returns a signature attesting to the liveliness (step 6.). It is worth noting that the relying party cannot link the mediator's signature to any particular eID because the signed message depends on its challenge and a hash value that is randomized using a 128-bit nonce. The FIDO-AC application generates a proof, which proves the knowledge of the hash preimage (i.e., DG data) and that this personal data is according to the policy Policy_S (e.g., above 18 years old). Both (att_m, σ_m) and π_{zkp} are sent back to the FIDO client (step 7.) and attached to the FIDO2 transaction (step 8.). Finally, the FIDO client sends the complete assertion to the relying party, which runs the signature and ZKP verification.

B FIDO-AC Implementation elements

FIDO-AC Application with Mediator: The application communicates with ePassport through an NFC interface with active NFC scanning triggered by user interaction. This approach enables the mobile application to avoid spawning unnecessary FIDO-AC processes on accidental proximity triggers of NFC events. The ePassport employs a password-based key-established protocol to protect the communication between the eID and the terminal. Therefore, our FIDO-AC application implements the PACE protocol to establish this secure channel. We also implemented the alternative BAC protocol that served as the backup for backward compatibility if the eID does not support PACE. PACE and BAC require a password as input, composed of user data such as document number, expiry date, and date of birth. Manual entry of the data is tedious and error-prone. The mobile application provides an optional optical character recognition (OCR) to scan, read and parse the necessary information on the document's machine readable zone (MRZ). Furthermore, the read data can be cached if the user opts in. Caching the personal data reduces the eID read time, which initially takes $\sim 1060ms$. Moreover, the cached data can be encrypted at rest to provide further protection (defense-in-depth), albeit at a slightly higher computational cost.

The mobile application also displays information about the origin of the FIDO-AC requester and its queries. It allows users to check whether they are expecting the origin and the associated queries. Before the eID is scanned, the user retains the right to cancel the transaction and downgrade to FIDO.

Once the reading of the eID is done, the FIDO-AC mobile application will communicate with a mediator as described in Section 5.2. In our prototype implementation, we opted for a local mediator implemented as part of the mobile application. This approach relies on the property that (for a secure system) hardware-backed keys can only be used by the application that generated it, and the TEE enforces such a boundary. To attest to an honest computation, the FIDO-AC mediator will use a package-bound hardware-attested key to sign the result described in Section 5.1. The relying party verifies the signature and is assured about the honest behavior of the mediator component.

The zero-knowledge proof system selected to realize the privacy-preserving functionality is Groth'16 ZK-SNARK

[19], mainly for its efficiency, short proof size, and the availability of existing implementation, rust-arkwork [12] and libsnark [26]. In this case, we chose to use rust-arkwork [12]. One of the additional deciding factors in choosing this library was the ease of cross compilation between ARM and x86. In FIDO-AC, we prove statements that use such examples as building blocks. Unfortunately, Groth'16 ZK-SNARK requires a trusted setup to generate a common reference string (CRS) for proving and verifying. The FIDO-AC server will be trusted to host an honestly generated CRS repository. We do not introduce new trust assumptions here since we already trust the FIDO-AC server to provide the certified FIDO-AC mobile application needed to secure the user-side mediator implementation.

During verification, the verification service checks the correct FIDO-AC response and whether the mediator's public key has a valid hardware-backed key attestation. A valid hardware-backed key attestation for this scenario requires the presence of the mediator package name, the mediator package's certificate fingerprint, and an attestation challenge that is the same as the FIDO challenge. Using this challenge, we bound the mediator's hardware-backed key to the particular FIDO session, which is supported thanks to the functionality provided by the Android API. It is possible to have a more robust integrity check utilized by PlayIntegrity API [18]. However, this particular approach is not considered for implementation because of the reliance on the GooglePlay service that might not be available for some Android devices.

FIDO-AC Server: One of our main goals in designing the FIDO-AC systems is to reduce the complexity of the deployment process. To meet this goal, we particularly focused on the design of the FIDO-AC Server. We prepared the Server as an independent element of architecture (i.e., implemented as a self-contained and stateless docker container). Notably, the Server is technology agnostic as it publishes a REST interface over HTTP. The main tasks of the Server are the following: distribution of the common reference string for the zero-knowledge proof system, verification of the mediator's attestation, and the zero-knowledge proof created by the FIDO-AC application.

The ZKP trusted setup parameters generated by the system must be propagated to all parties using the proof system (i.e., prover and verifier). A naive method would be to include the parameters in the applications (e.g., in the configuration files). Even though this could work well with our FIDO-AC mobile application, integration on the verifier side could raise usability issues as it requires integration with a relying party (usually out of the FIDO-AC system control). Considering the learned lessons from the configuration problems of other complex security protocols (e.g., the TLS configuration issues reported by Krombholz et al. [25]), we decided to externalize and automate the parameter configuration process. Therefore, we modeled the FIDO-AC Server as a centralized repository for CRS data, which can be conveniently discovered using a single HTTP call.

The optional functionalities of the FIDO-AC Server are introduced to minimize the integration efforts. Notably, the proof verification functionality can be implemented as a local module (i.e., in the relying party). However, this approach does not scale well, as the great diversity of technologies used for commercial applications makes a single implementation of the verifier impossible. Therefore, we decided to encapsulate and extract this functionality to a separate component (i.e., FIDO-AC Server) which can be either local for the relying party (e.g., deployed next to the application) or hosted by an external trusted party. Similarly to the trusted setup functionality, the verifier can be reached by sending a simple HTTP request. Following our approach, the relying party can integrate with only marginal changes to its source code (i.e., one HTTP call).

FIDO2 Integration: As discussed in Section 5.3, implementing a fully functional FIDO2 extension is significantly limited. Therefore, the processing of the *fidoac* extension is implemented before and after WebAuthn API calls. We use the relying parties' challenge to bind the extension data to the FIDO assertion. Below, we briefly describe the FIDO2 flow enhanced with *fidoac* extension. The steps of the process are depicted in Figure 6.

The flow starts with generating a FIDO assertion request with a *fidoac* extension (step 1.). The processing of the *fidoac* extension is encapsulated inside *fidoac.js*, and thus it does not require any change of the existing FIDO JavaScript code. To achieve a frictionless integration, *fidoac.js* overwrites the original *navigator.credentials.get* function with a custom implementation (steps 2. and 3.). In consequence, *fidoac.js* can preprocess FIDO assertion and forward it to WebAuthn API (i.e., the original *navigator.credentials.get* function). The processing of the *fidoac* extension involves communication with the FIDO-AC service (steps 4. and 5.) using internal (i.e., localhost) HTTP calls. The binding between FIDO-AC data and FIDO transaction is done via appending the SHA-256 hash of data to the FIDO assertion challenge (step 6.). The modified request is passed to WebAuthn API (steps 7-10) to generate a signed assertion. Notably, our *fidoac* extension is not forwarded to the FIDO authenticator due to the web browser limitations, and thus the only signed modification is the appended part of the challenge. The response from WebAuthn API is again intercepted by *fidoac.js* and enriched with fidoac data (inside the *clientExtensionResults* element). The final response is sent back to the server (steps 11. and 12.). Because the challenge was modified, the FIDO server has to execute the same action (i.e., hashing the extension data and appending it to the challenge) to verify the FIDO2 transaction successfully.

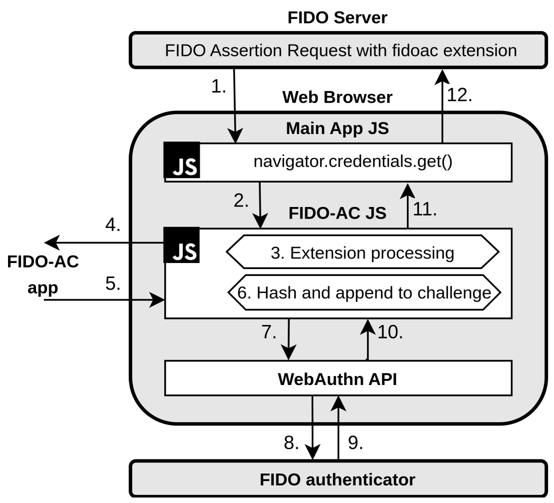

Figure 6: FIDO-AC extension processing

C FIDO2 challenge

The FIDO2 protocol uses challenges to prevent replay attacks. In the FIDO-AC system, we use this mechanism as a binding mechanism (see Section 5.3). Even though theoretically (i.e., in the WebAuthn specification), the length of the challenge is not limited, the software implementations might trigger errors if the challenge is above a certain threshold. We studied the documentation of various FIDO2 authenticators, and we could not find any explicit limitations, which suggests that vendors do not artificially limit the challenge size and, thus, are compliant with the vast majority of the FIDO Servers.

We empirically tested various authenticators to ensure their compatibility with the FIDO-AC system. In our test, we examined both platform and roaming authenticators using the Chrome browser on mobile. We tested Android and iOS platform authenticators and Yubico 5 roaming authenticators. The test procedure repeatedly triggers FIDO2 authentication, increasing the challenge size before each run. The evaluation stops when the threshold of 100Kb for the HTTP message is reached. We adopted this threshold from one of the HTTP servers (i.e., expressjs). The results confirm that authenticators allow significantly longer challenges. Therefore, we claim that the majority of commercial authenticators should support our approach of appending the hash value (256-bits of SHA-256).

D FIDO Extension Considerations

The WebAuthn API implementations are known not to support custom extensions, and thus various solutions have been proposed. We analyzed academic and industry approaches and identified three ways to mitigate this problem. The intuitive one is bypassing client limitations by implementing a custom FIDO client and authenticator. For example, Gou et al. [20] follow this approach to introduce a QR-based registration flow. Unfortunately, in our case, replacing popular FIDO clients and authenticators is not possible because of requirement **R.5** and requirement **R.6**. Okawa et al. [27] presented a solution closer to our needs, which used relying party code to implement WebAuthn API. It is a smart way to evaluate custom FIDO2 solutions. However, it is not suitable for production deployment. A solution that could be used in the wild is the one proposed by Putz et al. [29]. The authors solve the extension issue by implementing a plugin for web browsers (FIDO clients) that passes the extension content to the authenticator. Even though this approach solves the limited FIDO clients, it does not address requirement **R.5**. For example, in the mobile use case, browsers limit or forbid extensions, which makes integration difficult. Additionally, the arbitrary plugin raises adaptation and scalability challenges (requirements **R.5** and **R.6**) as it needs to be introduced for each user separately.