FeIDo: Recoverable FIDO2 Tokens Using Electronic IDs

Solving Token Loss and User Data Privacy via TEE-protected Attribute-based Credentials

Fabian Schwarz*
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
fabian.fsblack@gmail.com

Khue Do*
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
khue.do@cispa.de

Gunnar Heide
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
gunnar.heide@cispa.de

Lucjan Hanzlik
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
hanzlik@cispa.de

Christian Rossow
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
rossow@cispa.de

ABSTRACT

Two-factor authentication (2FA) mitigates the security risks of passwords as sole authentication factor. FIDO2—the *de facto* standard for interoperable web authentication—leverages strong, hardware-backed second factors. However, practical challenges hinder wider FIDO2 user adoption for 2FA tokens, such as the extra costs ($20-$30 per token) or the risk of inaccessible accounts upon token loss/theft.

To tackle the above challenges, we propose FeIDo, a *virtual* FIDO2 token that combines the security and interoperability of FIDO2 2FA authentication with the prevalence of existing eIDs (e.g., electronic passports). Our core idea is to derive FIDO2 credentials based on personally-identifying and verifiable attributes—name, date of birth, and place of birth—that we obtain from the user's eID. As these attributes do not change even for refreshed eID documents, the credentials "survive" token loss. Even though FeIDo operates on privacy-critical data, all personal data and resulting FIDO2 credentials stay unlinkable, are never leaked to third parties, and are securely managed in attestable hardware containers (e.g., SGX enclaves). In contrast to existing FIDO2 tokens, FeIDo can also derive and share verifiable meta attributes (*anonymous credentials*) with web services. These enable verified but pseudonymous user checks, e.g., for age verification (e.g., "is adult").

CCS CONCEPTS

• **Security and privacy → Multi-factor authentication**; *Trusted computing*; *Web protocol security*.

KEYWORDS

FIDO2; eID; token loss; authentication; SGX; anonymous credentials

1 INTRODUCTION

Passwords still represent the most popular type of credentials in web authentication—despite their widely-studied deficiencies [27], such as the risks of password database breaches [49], shoulder surfing [25], phishing [45], or low entropy passwords [22]. To mitigate these issues, a growing number of web services offer *two-factor authentication* (2FA) to increase authentication security. 2FA schemes usually ask the user for some proof of possession, such as one-time passwords (OTP) sent to the user's mobile phone. However, common second factors, e.g., SMS OTP [38] or OTP apps, are vulnerable to client-side and server-side leaks of the OTPs and their secret seeds. Furthermore, they require interceptable user input for entering the OTPs, and users must manage them for each service and client device. To enable stronger 2FA, the FIDO2 standard defines how to use protected hardware tokens (also called *authenticators*), especially for web authentication [41]. FIDO2 follows a challenge-response protocol where for each origin (e.g., web service), the hardware token securely generates and stores a public key pair. Upon authentication, the FIDO2 client first verifies the server origin, after which the token digitally signs a server-chosen challenge using its private key, which is never accessible outside the token.

Given the above security benefits, FIDO2 has become the *de facto* standard for strong, interoperable authentication supported by many popular services. However, while FIDO2 *support* is increasing, the actual *user adoption*—i.e., users leveraging FIDO2-compatible hardware tokens—lags behind. Two fundamental downsides of hardware tokens hinder a wider adoption. **(1) Costs:** Users are reluctant to buy dedicated hardware tokens as they incur extra costs. Even basic FIDO2 tokens cost around $20-$30, which is a non-negligible investment. **(2) Token loss:** Hardware tokens are subject to loss or theft. Users might no longer be able to log in, as token-based credentials cannot be backed up. Indeed, token vendors recommend registering at least two different tokens to the same user account [54]—further increasing the costs and hampering the usability of hardware tokens overall. Alternatively, users must fall back to less secure alternative authentication or account recovery schemes (e.g., recovery codes) if provided by the web service [52].

The barrier of additional costs has motivated vendors to offer FIDO2-compatible "virtual" tokens. These tokens do not require extra hardware but root their signing security in trusted hardware of client devices. For instance, Android and Windows 10 have FIDO2-certified authenticators: Android's *Keystore* backed by ARM TrustZone [6], or the *Windows Hello* authentication service relying on the Trusted Platform Module (TPM). On the one hand, this allows virtual tokens to securely store authentication-relevant secrets in trusted hardware so attackers cannot steal them. On the other, this advantage comes at the cost that users can still not back up their

*We thank Saarland University for supporting Fabian Schwarz and Khue Do.

authentication secrets. Furthermore, not all user devices offer the trusted hardware (e.g., TPM) required by the virtual FIDO2 tokens.

In this paper, we propose FeIDo, a fully FIDO2-compliant virtual token that tackles the challenges of costs *and* token loss. FeIDo is mainly designed for—but *not* limited to—providing private users with a strong second factor for two-factor web authentication. FeIDo is a virtual FIDO2 token utilizing electronic identifications (eIDs) such as electronic passports or ID cards. FeIDo uses the communication interface of eIDs as standardized by the International Civil Aviation Organization (ICAO) to extract personal data from the documents. eIDs can prove the authenticity of personal attributes, such as the user's name, place of birth, and date of birth. These personal attributes then form the basis for FeIDo's user authentication. eIDs nicely address the above authentication challenges in FIDO2: **(1) No extra costs:** eIDs obsolete the need for dedicated security tokens for eID holders. Over 1 billion citizen already own electronic IDs or passports [3, 5] (cf. § 2.2). Our setup thus does not impose additional hardware requirements—users can leverage eID readers such as NFC readers that ship with off-the-shelf phones (cf. § 2.2). **(2) Token recovery:** The authentication in FeIDo is not bound to a particular eID but only to its verifiable personal attributes. These attributes do not change even if a lost/stolen eID is replaced by a new one, which enables direct credential and thus account recovery.

While personal attributes could be directly shared with and verified by authenticating services, most attributes stored on eIDs are privacy-sensitive (e.g., name, place of living) and *not* required for authentication. There are many cases in which users wish to remain pseudonymous, such as in adult websites, forums (e.g., health forums), learning platforms for kids, or even social media. Therefore, we design a FIDO2-compatible attribute-based authentication scheme in which third parties do *not* learn personal details and credentials are unlinkable. Our core idea is to use trusted remote *credential services* that validate and vet—but never share or leak—the personal data. The credential services feed the personal user data as input to a key derivation function to derive *attribute-based credentials*. The resulting credentials depend on personal attributes (name, date of birth, and place of birth) and a secret chosen by the credential services but are unlinkable. To guarantee that the credential services can be trusted to protect personal data against third parties (including the hosting providers), they execute in attestable Trusted Execution Environments (TEEs). Users can remotely attest the protection and authenticity of a given credential service, verifying its validity before sharing their personal data.

In addition, our design comes with an attractive extension that is not in the scope of existing FIDO2 tokens. FeIDo enables *anonymous credentials* that allow web service providers to learn pseudonymous meta user attributes and verify their authenticity. For example, adult websites may have to ensure that their users are of legal age, or governmental websites may want to restrict services to residents. FeIDo's credential service derives such meta attributes (e.g., "is adult") from the raw eID data (e.g., *dob* = May 14, 1981) in an attestable way and, at the same time, guarantees that the raw user data is never exposed outside of the credential service's TEE.

Our design on the user side is agnostic to the choice of a concrete OS, hardware, and eID. Users can use standard interfaces such as NFC to read personal data from their eID and prove its authenticity to a credential service (cf. § 2.2). The credential service requires a TEE providing remote attestation, data encryption, and integrity checks, e.g., as available on public cloud platforms—and one instance can easily handle tens of thousands of users. FeIDo clients perform remote attestation before forwarding data to the credential service—notably *without* requiring TEE support themselves. Clients also do not need to back up their credentials to withstand device loss, as credentials are always freshly derived from eID data. FIDO2-capable web services can readily use FeIDo-backed credentials, and can leverage anonymous credentials using FIDO2 extension fields. In our evaluation, we instantiated this general design in a concrete setting without losing generality. We securely implement an open source[1] prototype that consists of an Android app reading personal data from an ePassport and a TEE-protected credential service receiving and vetting this personal data to derive signing keys for authentication. We evaluate our prototype by measuring the FIDO2 authentication duration on the well-known *webauthn.io* test page [37]. We show that FeIDo is comparable in efficiency to existing FIDO2 hardware tokens while tackling their shortcomings. In summary, we make the following contributions:

- We design FeIDo, a virtual FIDO2 token that enables account recovery on a token loss and is readily available without extra costs to the large population of users owning a compliant eID such as ICAO-standardized ePassports.
- As the crucial enabler for account recovery, we present the concept of *attribute-based* FIDO2 credentials, protecting personal data within an attestable TEE.
- FeIDo enables anonymous credentials (e.g., age group) that web providers can verify without having access to privacy-infringing raw user attributes (e.g., date of birth).
- We analyze the security and prototype[1] of FeIDo.

2 BACKGROUND AND RELATED WORK

We first provide the reader with background information on FIDO2 and electronic IDs and describe their current shortcomings.

2.1 FIDO2

The FIDO2 standard (and its predecessor U2F) describes how to use hardware tokens for authentication. Such tokens rely on public-key cryptography (PKC). To form authentication credentials, the client creates a dedicated key pair per service as credentials with the help of a hardware token ("*authenticator*"). The hardware token generates and stores these key pairs. To register their credentials with an account, users send their service-specific public key to the authenticating server ("*relying party*"). The corresponding private key never leaves the hardware token, even if the user's system ("*agent*") is compromised. To authenticate a user, client and server follow a challenge-response protocol, as shown in Figure 1 for a web authentication setting. The client signs a server-chosen challenge using their authenticator (token). To this end, the agent verifies the origin (i.e., domain and port number) of the relying party (web service). The authenticator then uses the origin-specific private key to sign the challenge. The relying party uses the public key associated with the authenticating account to verify the signed

[1] Prototype available at: https://github.com/feido-token

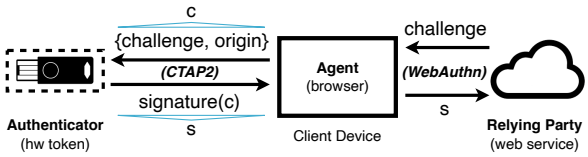

Figure 1: Simplified flow of a FIDO2 web authentication.

client response. In essence, possession of the private key serves as proof for authentication.

This authentication workflow is realized in two core FIDO2 components, *WebAuthn* and *CTAP2*. WebAuthn [41] standardizes the interface of PKC-based authentication on the web. Basically, WebAuthn defines the communication between the client device and the web server, including the challenge-response protocol mentioned above. The standard defines an API for the agents that defines how to generate fresh credentials (*MakeCredential*) and get the response to the relying party's challenge (*GetAssertion*). The Client to Authenticator Protocol (CTAP2) complements WebAuthn. CTAP2 defines the communication between the authenticator and the agent.

The FIDO2 standard supports using authenticators in several private and enterprise use cases for secure authentication [10]. In this work, we set the prime focus on FIDO2 authenticators as secure second factors for web authentication by private users. However, we will later explain how our proposed scheme can be used as a single factor (§ 7.1) or in enterprise-focused use cases (§ 7.2).

2.1.1 Open Challenges for FIDO2. FIDO2 mitigates password cracking via database breaches, phishing attacks, and secure storage of credentials. FIDO2 also makes users unlinkable across services, as the authenticator generates a fresh public key for every relying party. However, FIDO2 introduces a new set of problems.

Costs: The extra costs for dedicated hardware are one of the main obstacles to scaling FIDO2 to the masses. Even the most basic hardware-based authenticators cost around 20-30 USD, a non-negligible expense for many. To tackle this challenge, vendors offer FIDO2-compliant "virtual" authenticators that use trusted computing features (e.g., TPM or secure enclaves) of the user's hardware. Google introduced an API to Android OS that allows the user to store FIDO2 keys inside the ARM TrustZone-backed Android Keystore [6]. To use this virtual FIDO2 authenticator, the user relies on the authentication mechanism used to access the phone, i.e., a PIN code or biometric features. The Windows Hello authentication service provides a similar solution based on TPMs. FIDO2 is also available for macOS and iOS using Apple's secure enclave for key management and FaceID and TouchID as user access control [14]. Chakraborty and Bugiel proposed an authenticator called *simFIDO* [15] based on simTPM [16]. They use a cheap SIM card running the Java Card OS as TPM to implement a FIDO2-compliant authenticator, mitigating the cost issue. On the downside, this solution requires mobile service providers to support and install extra applications (simFIDO and simTPM) on their SIM cards. While all these approaches mitigate the cost issue, they require special trusted computing features on the client side, which are not present in many consumer devices. Furthermore, unless users choose alternative authentication methods, they will lose access to their credentials if the device gets lost or stolen, as discussed next.

Token Loss: Another open challenge of FIDO2 is to provide a cost-efficient and practical solution for account recovery in case the authenticator is lost. In principle, as the private credential keys are protected on the authenticator and can not be extracted, losing an authenticator implies losing access to all authenticator-protected accounts. The naive yet recommended solution [52, 53] is registering an additional authenticator to the same account as a backup, doubling the costs. Alternatively, if supported by the relying party, additional authentication factors or service-specific recovery strategies, such as SMS OTP or recovery codes, can be used, which, however, degrade security and face several drawbacks [35, 38].

2.2 eIDs for Authentication

This paper proposes a FIDO2-compatible virtual authenticator that uses electronic IDs (eIDs). eIDs and their user-based associations are inherently well-suited for authentication. A combination of personal data such as the full name, day of birth, and place of birth is reasonably specific to be used in strong authentication mechanisms. For many eIDs, such personal data can be (i) securely read electronically, (ii) verified to stem from a valid eID, and (iii) accompanied by freshness guarantees that show if a user currently possesses the eID. eIDs typically include RFID chips with NFC interfaces, for which respective NFC support is widely available on consumer phones, including all iPhones released since 2016 (iOS $\geq$ 13.2) [32, 40] and most Android phones since '15, e.g., from LG and HTC, all Samsung phones since '15, and all Huawei phones since '17 [32, 39]. To prevent eID spoofing and ensure data authenticity, the personal data is signed by a trusted issuing authority, e.g., a state. Furthermore, eIDs usually physically protect memory for secret storage and prevent unauthorized access and detect eID cloning.

The eID interfaces are either standardized (in the case of electronic passports) or subject to the issuing country (in the case of national IDs). Having said this, an ever-increasing number of countries deploy national eIDs or ePassports that provide the above capabilities. For example, every eID following the ICAO standard [48] for electronic machine-readable travel documents (eMRTDs) satisfies these requirements. As of mid-2019, over 150 countries issued ePassports [5], including the top 10 most populated countries constituting more than half of the earth's population. Moreover, also many national IDs implement this standard. For example, starting from August 2021, all European Union member states are required to issue ICAO-compliant national ID cards [1]. Compliant IDs are also provided by Panama, Uruguay, Algeria, Saudi Arabia, Ukraine, Kyrgyzstan, and mainland China ID cards for Hong Kong, Macau, and Taiwan [2]. Several other national eIDs implement similar interfaces, often defined in regional or country-specific standards [3]. Taking into account the accessibility of ICAO-compliant eIDs throughout the paper, without losing generality, we assume the eID is an eMRTD. This way, we can explain the general concept by the example of concrete communication protocols.

An eMRTD interacts with a client through an NFC reader via well-defined protocols. Three of these protocols [47], which map to the eID capabilities mentioned before, are relevant in our context:

(i) PACE: Password Authenticated Connection Establishment (PACE) is a password-based access protocol to protect the communication between an eMRTD and a reader. PACE assures that the reader is authorized to access certain data groups on the eID [7, 13]. The chip asks the reader for a (static) password as access control, typically printed on the eID. The user provides this eID-specific secret to the reader. This password can optionally be cached in software to ease future accesses. The chip and reader derive a Diffie-Hellman session key from this weak password to grant/obtain read access and to establish a secure channel for subsequent data exchange.

(ii) PA: During Passive Authentication (PA), the reader verifies the data authenticity of the retrieved personal data. Next to raw data, an eMRTD chip can also send a document security object (DSO), i.e., signed hash values of the personal data stored on the eID. The DSO is also signed and can be verified, ensuring that only a trusted eID authority can create valid DSOs. Readers can therefore validate the eMRTD data by comparing data hashes.

(iii) CA: While PA ensures data authenticity, Chip Authentication (CA) prevents data and eID cloning. When initiating CA, the chip shares a static public key with the reader. By checking this public against the previously hash-protected eID data received via PACE, one can verify the binding between the present eID and user data. In addition, a successful CA channel establishment proves that the eID is in possession of the *unclonable* CA private key and therefore authentic. After the reader sends an ephemeral public key to the chip, both the chip and the reader derive a shared key between the two parties from these keys—technically, a Diffie-Hellman key exchange. They later use this shared key to derive session keys (encryption key and MAC key) for secure communication.

2.2.1 Risks of eID-Based Authentication. The previous discussion showed that eIDs provide well-defined and easy-to-access interfaces to extract verifiable personal data securely. However, while exposing eID data to third parties for authentication is technically possible, it infringes a user's privacy as the personal data stored on eIDs is highly privacy-sensitive (e.g., date of birth, place of living). The data goes much beyond what is necessary for authentication purposes. Furthermore, there are many use cases in which users wish to remain pseudonymous (e.g., health forums), as described in § 1. Therefore, we see the potential for an eID-based authentication scheme that can be based on an eID's verified user data *without* revealing any user data to the authenticating service.

State authorities already identified authentication as another eID use case. Some proposals even make eID-based identification compatible across countries, such as eIDAS in the EU [8, 28, 43]. However, they either infringe privacy per the above argument or rely on a country-specific pseudonym functionality whose authentication credentials are invalidated every time an eID is renewed.

3 FEIDO: DESIGN GOALS AND THREATS

In order to overcome the roadblocks of FIDO2 adoption, this work aims to design a new authenticator, called *FeIDo*. In the following, we present FeIDo's goals and threat model.

3.1 Goals and Requirements

The goal of FeIDo is to form a FIDO2 authenticator for secure web authentication, which overcomes current limitations of hardware and virtual authenticators. We design FeIDo to combine concepts of virtual FIDO2 authenticators with that of smartcards owned by billions [3] of citizens: eIDs such as electronic passports or ID cards. To guide the design of FeIDo, we have defined six requirements:

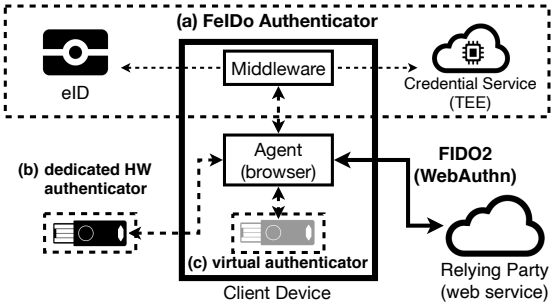

Figure 2: Comparison between (a) FeIDo and (b) existing hardware and (c) virtual FIDO2 authenticators.

R1 Compatibility FeIDo must be compatible with the standard FIDO2 protocols for authenticator-based web authentication.

R2 Economic FeIDo must build on commodity hardware and eIDs that are readily available for eID holders *without* extra costs.

R3 Account Recovery The user must be able to recover access to their accounts on a FeIDo authenticator loss *without* having to register additional authenticators.

R4 User Privacy Third parties (incl. relying party, FeIDo provider) must neither be able to access personal user data directly nor link personal data to FeIDo's web credentials.

R5 Platform Independence FeIDo must not bind users to a specific software or hardware vendor (e.g., OS, client device).

R6 Anonymous Credentials The relying parties should be able to learn authenticated yet pseudonymous meta information on a user for verification, but *without* violating user privacy.

FeIDo provides a strong FIDO2 second factor for private users' web authentication (2FA). However, FeIDo is *not* limited to this application and can support additional FIDO2 use cases, such as passwordless and enterprise schemes discussed in § 7.1 and § 7.2.

3.2 Threat Model

We follow the threat model of FIDO2 web authentication [9] and extend it to include FeIDo-specific components and entities. Analogous to the FIDO2 setting, a trusted user wants to register and log into a web service via their browser using the FIDO2 protocols and the FeIDo authenticator. We trust the web service and assume a secure, authenticated connection between the browser and service. From a FIDO2 perspective, the user's browser forms the *agent* while the web service forms the *relying party*. We assume the agent and client device to be trusted—following the trust setting of token-based authentication. While we could relax this assumption using TEEs on client devices, we also want to support users *without* access to special features such as Android's protected confirmation [31].

Figure 2 shows that FeIDo adds new components not present in existing authenticators: a user eID, a remote credential service, and a client-located middleware. We consider several attacks against each

of the three components and active and passive network attackers targeting the communication between them. We trust the individual components in the following ways: **eID:** Similar to hardware authenticators, we trust the eID's hardware-based protections and security protocols (§ 2.2) and the issuing document authority such that we can verify if a user's eID is genuine. That way, FeIDo can rule out attackers trying to spoof or clone arbitrary eIDs that impersonate others. We assume an attacker trying to steal and abuse the user eID for account hijacking (see § 5.3). Finally, we discuss the security of eIDs and explain the impact of malicious authorities on FeIDo in § 5.2.2. **Credential Service:** Service instances run on untrusted remote systems such as public cloud platforms. We assume that remote attackers, including the hosting providers and platforms, try to manipulate, steal, or clone credential services. Therefore, we operate each service in a secure, hardware-based container [44] protecting them against the above threats. Furthermore, we assume the containers support remote attestation to rule out attacks that impersonate the service. **Client Middleware:** We do not trust the middleware to perform any eID validation but merely use it as a proxy between the eID and the credential service. Following our assumption that we trust the client device, the middleware shares eID data only with a valid, protected credential service, verified via remote attestation.

We assume FeIDo's components to be free of exploitable vulnerabilities. To prevent physical and micro-architectural side-channels, we rely on existing smartcard features such as memory and constant-time cryptographic operations, similar to existing FIDO2 authenticators. We assume orthogonal defenses for the hardware-based containers protecting the credential services [11, 23, 46].

4 FEIDO: CONCEPTS AND DESIGN

In the following sections, we explain the concepts and design of the FeIDo authenticator, including its optional support for anonymous credentials. The section then concludes with a deployment analysis. We refer to FeIDo's design requirements (§ 3.1) in relevant passages.

4.1 Big Picture

FeIDo's core idea is to derive FIDO2 web credentials based on a user's unique, personal attributes that FeIDo securely retrieves from the user's eID (*R1*). At its core, FeIDo foresees hardware-protected and attestable *credential services* that convert privacy-critical personal attributes—name, place of birth, and day of birth—to pseudonymous authentication credentials. The credential service acts as a sort of "pseudonymizing proxy" that ensures unlinkability and preserves user privacy against both the authenticating services and the untrusted credential service hosters. FeIDo combines the advantages of token-based authentication (strong security and privacy) and eID-based authentication (easy token recovery, no extra costs), tackling the open challenges of both schemes (cf. § 2).

In FIDO2 terminology, FeIDo forms a virtual FIDO2 authenticator that combines TEE-protected credential services with user-owned eIDs. The credential service derives *attribute-based credentials* that are user-specific yet unlinkable (*R4*) and depend on a user's attributes rather than a specific physical device or eID. These attribute-based credentials have clear advantages when it comes to account recovery because any user eID carrying the *same* personal attributes can serve as a replacement eID (*R2+3*). In order to retain user privacy and credential security, each credential service instance is protected in a remotely verifiable hardware TEE and prevents attribute and credential leakage to third parties (*R4*). In addition, the service supports so-called *anonymous credentials* (*R6*), an extension that allows a relying party to learn meta user attributes, e.g., for age verification, *without* violating a user's privacy (see § 4.4)—a feature not present in current eIDs and authenticators. FeIDo's service-based design requires neither secrets nor trusted hardware features on client devices, which benefits account recovery, user costs, and cross-platform support (*R2+3, R5*).

Table 1: Feature comparison between FeIDo, existing hardware and virtual FIDO2 authenticators, and eIDs schemes.

	FeIDo	HW Aut	Virt. Aut	eID
R1 - FIDO2 Compatible	✔	✔	✔	×
R2 - *No* Extra Costs	✔	×	✔	✔
R2 - Widely Deployed	✔	×	✔	✔
R3 - Device Loss Rcvry	✔	✔	×	✔
R3 - Token Loss Rcvry	✔	(✔)	(✔)	✔
R4 - User Privacy	✔	✔	✔	×
R5 - Cross Platform	✔	✔	×	✔
R6 - Anon. Creds	✔	×	×	×

Figure 2 shows how FeIDo forms the virtual FIDO2 authenticator by combining an off-the-shelf user eID with two new software components: a TEE-protected remote *credential service* and a client-located *middleware*. The client middleware is a *secretless* component that interfaces the FeIDo authenticator with the FIDO2 agent (typically part of the browser) and securely mediates internal communication between the eID and credential service (see § 4.5).

4.2 Comparison to Existing Authenticators

FeIDo combines concepts of virtual FIDO2 authenticators with eIDs to overcome the drawbacks of existing hardware and virtual authenticators and enable new features, as shown in Table 1. FeIDo removes the need (and costs) for buying extra hardware authenticators dedicated "only" to authentication purposes. Instead, FeIDo can leverage the wide deployment of eIDs across billions of citizens and their compatibility with off-the-shelf phones (see § 2.2). In contrast, hardware authenticators have only limited user adoption due to their risk of token loss and their incurring extra costs (§ 2.1.1). In contrast to virtual authenticators, FeIDo relies on secure hardware containers on the credential service side rather than client devices. The credential services can use TEEs widely available on public cloud platforms (e.g., Intel SGX, AMD SEV-SNP [44]) and are thus easily shareable by thousands of users with negligible costs, similar to Tor nodes [24]. FeIDo poses no requirements on client devices and uses the client middleware only as a *secretless* proxy between eID and credential service. Therefore, similar to hardware authenticators, FeIDo is independent of the specific client device, which enables easy porting of the middleware to other client platforms. Furthermore, a client device loss does not affect FeIDo or hardware authenticators, while virtual authenticators and their credentials are tightly bound to a specific client platform and device.

FeIDo retains account access on a token loss because any replacement eID carrying the same user attributes enables access to a user's credentials and thus accounts (§ 4.3). Therefore, in contrast to existing authenticators, FeIDo can recover from a token loss without requiring backup tokens (cf. hardware authenticators), insecure vendor-specific cloud backups of token secrets (cf. some virtual authenticators), or insecure service-specific recovery methods (e.g., recovery codes). A potential downside can be a longer renewal time for eIDs (a few weeks) compared to buying a new hardware authenticator if no replacement eID is available (e.g., driver's license, ePassport). Finally, FeIDo enables anonymous credentials during the authentication process, e.g., for pseudonymous age verification—in contrast to existing FIDO2 authenticators.

Finally, when comparing FeIDo to naïve eID-based authentication *without* the proposed middleware and credential services (Table 1, last column), we see that they are not FIDO2 compatible and fully violate a user's privacy by exposing personal data to the authenticating services, as described in § 2.2.1. Furthermore, they do not support anonymous credentials.

4.3 Attribute-based Credentials

FeIDo's credential service performs the actual FIDO2 *authenticator operations* for web registration and login, i.e., credential creation and assertion [41]. We have designed the resulting virtual token as a user-independent network service. This design mitigates the risk that users can lose secrets. Instead, the credentials become remotely usable for users from any device (*R2+3, R5*). The credential service keeps no persistent user or credential state. Instead, to distinguish users and bind their FIDO2 credentials exclusively to them, the credential service *dynamically derives* a user's credentials using a key derivation function (KDF). The unique, personal attributes from the eID are fed to the KDF, resulting in *attribute-based credentials*.

4.3.1 Reading eID User Attributes. When FeIDo authenticates a user, the credential service first securely reads a user's personal attributes from their eID. The service cooperates with the client middleware on each authentication request to remotely access the user's eID. By explicitly requiring a user's eID on each request, the credential service can verify the freshness of the attributes and eID and thus guarantee that credentials are only derived if the eID is valid and present. Otherwise, attackers could provide a different user's data for an impersonation attack or use bogus eID clones. To enable secure eID access and verification, FeIDo requires eIDs to provide support for (1) establishing a secure end-to-end connection to them and (2) verifying the authenticity and integrity of the eID and its stored user attributes. In our eMRTD-compatible prototype, the credential service retrieves user attributes—tunneled via the client middleware—leveraging the eID protocols described in § 2.2.

4.3.2 Credential Derivation. The credential service now feeds the personal user attributes into a KDF to derive user-specific credentials. The KDF input binds the resulting credentials to (a) the eID owner and (b) the relying party. The credential service receives the relying party's effective domain (or origin) rp_{id} from the client middleware. For the user binding, the credential service chooses a unique set of attributes from the personal user data included in eIDs: the user's full name (*name*), date of birth (*dob*) and place of birth (*pob*), and state (*state*) issuing the eIDs. For now, we assume that the above combination of user attributes (i) remains constant for a given user and (ii) provides sufficient uniqueness guarantees. We discuss these properties in more detail in § 4.6.2 and § 5.2.3, and explain the prevention of cross-state attacks in § 5.2.2.

To guarantee the security of the attribute-based user credentials, the KDF input additionally needs to include a secret s_{kdf} that is only accessible by a genuine, hardware-protected credential service. Lacking this secret, attackers cannot recalculate credentials purely based on the correct personal user attributes and relying party information (rp_{id}). Using the above inputs, we can now derive the credentials using a KDF. The credential service leverages an HMAC as the KDF, uses s_{kdf} as the HMAC's key [34], and all user attributes and relying party information as HMAC inputs. More formally:

$$h_{cred} = HMAC(s_{kdf}, rp_{id}|name|dob|pob|state) \quad (1)$$

where "|" is the byte string concatenation. The credential service interprets the resulting hash h_{cred} as the private key $\mathsf{sk}_{\mathsf{cred}}$ of the user's WebAuthn credential according to the specified signature algorithm and then calculates the respective public key $\mathsf{pk}_{\mathsf{cred}}$.

This reproducible public key pair can now be used in the FIDO2-typical way: During a credential registration (*MakeCredential*), the credential service includes the public key $\mathsf{pk}_{\mathsf{cred}}$ in the response. For a login (*GetAssertion*), the credential service uses the private key $\mathsf{sk}_{\mathsf{cred}}$ to sign the relying party's assertion challenge [41]. Personal data such as user attributes are *not* transmitted to the relying party. Furthermore, the credential service does neither persistently store user data nor credentials but instead deletes them at this point.

4.3.3 Cloud-Based Design. FeIDo foresees multiple redundant credential service instances that operate on public cloud platforms, which mainly serves two purposes. First, users do not risk losing the key derivation secret—all they need for authentication is stored on replaceable eIDs. Second, as public credential services can be hosted and shared by many, the costs become negligible. For a more detailed discussion on deployment scenarios, we refer to § 4.6.1.

Given that the credential service is offloaded to the cloud, users want to ensure that a contacted service instance is trustworthy, i.e., it is a genuine service that is protected and does not leak s_{kdf} or personal data. Furthermore, the communication between the credential service and the middleware must be secured. Therefore, each credential service instance operates in a TEE that allows achieving all these properties by offering integrity, attestation, and confidentiality. Several TEE implementations provide these principles [44], like Intel SGX, AMD SEV-SNP, or RISC-V Keystone—of which some are widely available on cloud platforms.

In our prototype, we have chosen an Intel SGX-based TEE. Intel SGX is a widespread, commodity server-grade CPU extension that provides user-level TEEs, so-called *enclaves*. While Intel has deprecated SGX for consumer CPUs, it continues to support SGX for cloud CPUs—exactly our setting. SGX enclaves run in dedicated, confidentiality- and integrity-protected memory regions and expose a minimal trusted computing base (TCB), including only their code, data, and the CPU [19]. Using Intel SGX's remote attestation, FeIDo users can verify the exact code and data running inside an

enclave based on hardware-issued, cryptographic proofs and bootstrap authenticated connections to them [33, 50]—*without* requiring access to SGX-capable hardware on the client side.

The TEE-based design is key to securing a cloud-based KDF and shielding the secret from attackers. The credential service randomly generates s_{kdf} on its *initial* startup. To be able to derive the same user credentials after a restart, the credential service must either previously use Intel SGX's sealing capability to store s_{kdf} on disk securely for recovery [19] or securely receive a copy by another credential service instance after mutual remote attestation (cf. § 4.6.3). Either way, the secret can never be read outside of an enclave.

4.4 Anonymous Credential Extension

The design of the credential service can be extended towards *anonymous credentials*—a feature neither provided by eID schemes nor FIDO2 authenticators (§ 4.2). In certain settings, the relying party might be interested in a curated form of user data. For example, the relying party may want to verify if users are adults or children (user age above/below X years) before granting them access to adult content or kids-only chats or may want to restrict its service to residents of a particular country. The credential service can derive such anonymous credentials from the trusted user attributes *without* leaking the raw data to the relying party (R6). Instead of receiving the raw user attributes (e.g., date of birth), the relying party *only* learns the anonymous credentials (e.g., "is adult"). In contrast to hardware-only tokens, FeIDo can add such meta attributes easily in software. In order to provide users full control over their data, the credential service shares only anonymous credentials that the user has explicitly permitted for a given relying party.

The relying party must be able to verify anonymous credentials. Otherwise, attackers could spoof meta attributes to bypass additional access policies, such as age restrictions. Therefore, the credential service, executing in a TEE, enables a relying party to remotely attest that a genuine credential service has generated the provided meta attributes (see § 4.5.4).

4.5 FIDO2 Integration

We now describe how our general idea of attribute-based credentials blends into the FIDO2 authentication workflow. The middleware mediates this integration as the central entity. In the following, we describe the middleware and explain the message flow to show how it ensures FeIDo's FIDO2 compatibility, how FeIDo can support the revocation of stolen (or lost) eIDs, and how it can extend the authentication process with anonymous credentials.

4.5.1 Client Middleware. The client middleware is the central communication mediator of FeIDo. It fulfills two roles regarding FIDO2: on the one hand, it serves as a WebAuthn/CTAP2 agent for the browser, which contacts the FeIDo authenticator and replies with WebAuthn responses for the relying party. On the other, the client middleware is part of the authenticator itself and mediates the communication between the eID, credential service, and optionally, the user. That way, the client middleware bridges the gap between multiple internal domains while ensuring seamless compatibility with the FIDO2 infrastructure. The middleware in our prototype uses an integration solution fully compatible with commodity browsers: a browser extension-based proxy agent. As shown in Figure 3, the

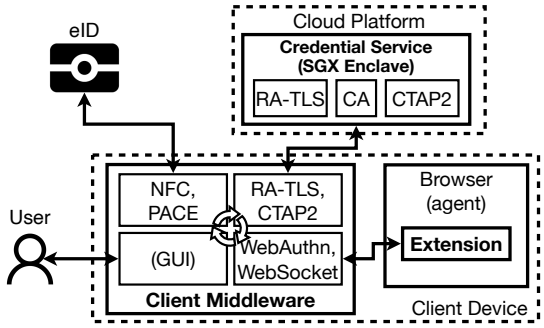

Figure 3: The client middleware is the central mediator and consists of multiple modules for communication with the browser extension, credential service, eID, and user.

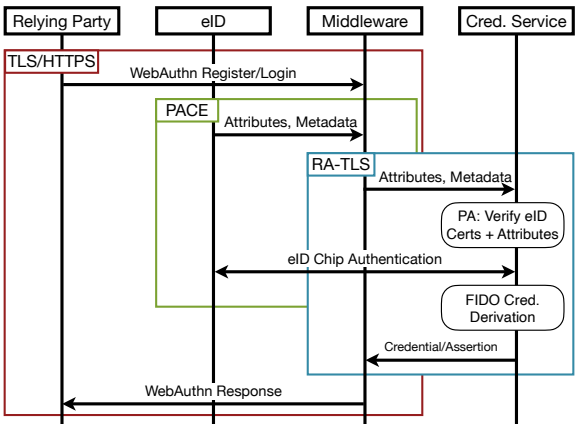

Figure 4: Message flow in a FeIDo authentication session.

browser extension cooperates with the client middleware to integrate FeIDo into a browser's WebAuthn framework.

4.5.2 FIDO2 Authentication Workflow. A FeIDo authentication session follows the message flow in Figure 4. On a web authentication request, the agent (browser) notifies the client middleware. The middleware then communicates with the credential service and eID via authenticated, end-to-end protected connections to come up with the respective WebAuthn credential (upon registration) or assertion response (upon authentication) for the relying party. The middleware mediates to give the credential service implicit access to the user's eID. We now iterate over these steps in more detail:

(1) FIDO2 Authentication Initialization. To initiate a FIDO2-based authentication, the relying party sends a WebAuthn registration or authentication request to the agent (typically, a user's browser). The agent then reaches out to the appropriate authenticator as described in § 2.1. In our prototype, the browser extension monitors browser sessions for calls to the WebAuthn request APIs. On a web authentication request, the browser extension notifies the client middleware via a local WebSocket connection and forwards the WebAuthn request. That way, the browser extension integrates the client middleware as a WebAuthn agent into the browser. The client middleware can then start processing the request as part of the

FeIDo authenticator. The choice of WebSocket has the advantage that it is compatible with commodity browsers and enables an easy relocation of the FeIDo authenticator to an auxiliary user device, e.g., a smartphone, in case the client device has no eID reader.

(2) Reading User Attributes. Next, the client middleware establishes a PACE-secured connection with the eID to read the personal attributes, according to the description in § 4.3.1. It defers the CA-based eID validity check to a later stage. Depending on the eID and agent, the client middleware may prompt for user interaction, such as putting the eID on the reader or entering its PIN, if required.

(3) Credential Service Interaction. The client middleware then establishes a secure network connection to the credential service to share the user attributes. To this end, the client middleware must verify that the connection is end-to-end protected and established with a genuine, TEE-protected credential service instance. Therefore, the client middleware uses RA-TLS [33, 50], which combines a TLS connection with a TEE-provided remote attestation. That way, the client middleware has a hardware-assured guarantee that it communicates with a credential service instance that is TEE-protected, well-behaving, and does not leak credentials or attributes to others.

As shown in Figure 3, the credential service incorporates an RA-TLS server endpoint [33] for attested, end-to-end protected communication with the client middleware. A TLS server key pair is freshly generated by the credential service enclave on startup or securely shared across different credential service instances (cf. § 4.6). RA-TLS binds the hash of the TLS server's public key to the SGX attestation report by incorporating it as authenticated user data and then integrates the resulting signed attestation quote inside the TLS server certificate. This integration enables the client middleware to verify the hardware-assured identity of the credential service and its binding to the established TLS connection as part of the TLS server certificate validation [33, 50]. The client middleware can then securely request WebAuthn operations from the credential service through the RA-TLS connection.

(4) eID and Attribute Validation. After receiving the user attributes, the credential service has to validate them using Passive Authentication (PA; cf. § 2). Furthermore, it uses Chip Authentication (CA; cf. § 2), initiated through the middleware's PACE channel, to verify that the service can access the eID that shipped the attributes. If both succeed, it uses the KDF to derive the user-specific authentication credentials: $\mathsf{sk}_{\mathsf{cred}}$ and $\mathsf{pk}_{\mathsf{cred}}$. Note that it is vital that the credential service performs this validation and only relies on the middleware to tunnel the respective communication. Otherwise, a malicious middleware could spoof arbitrary personal attributes and eIDs and thus perform impersonation attacks.

(5) Authentication Termination. Finally, to conclude the authentication, the credential service sends the reply for the requested authentication operation to the client middleware [41], i.e., the signed credential public key $\mathsf{pk}_{\mathsf{cred}}$ on a registration and the assertion challenge signed by the private key $\mathsf{sk}_{\mathsf{cred}}$ on a login request, respectively (see § 4.3.2). The middleware converts this into a *WebAuthn Response* and shares this with the relying party. To the relying party, this workflow is entirely transparent—it will not even notice that eID data was read/used at any point in time.

4.5.3 eID Revocation. FeIDo is designed as a *second* factor for 2FA. Therefore, stealing a user's eID does *not* suffice to gain access to the user's accounts (§ 5.3.2). Having said this, FeIDo supports revocation of stolen eIDs to prevent account hijacking attacks if attackers have *additionally* compromised the primary factor or if FeIDo serves as a sole authenticator (§ 7.1). FeIDo foresees three complementary eID revocation strategies, providing different tradeoffs.

First, the credential service can consult existing trusted databases for stolen eIDs on eID validation (§ 4.5.2, step 4), such as Interpol's I-Checkit service [30], and deny their use for authentication. The eID lookup does *not* leak personal user data as it only requires pseudonymous eID identifiers (number, type, issuing state). This approach does not add state or complexity to the credential service but requires trust in the service provider (e.g., Interpol) not to re-enable or DoS eIDs and might require a *small* fee [30].

Second, the credential service can derive and share *eID-specific* revocation certificates with the client middleware, which users can leverage on demand (e.g., upon eID loss) to denylist a particular eID for all credential service instances. The middleware can protect the certificates in client-side keystores that require physical device access and user authentication [6]. On eID validation, each credential service checks the denylist and aborts authentication. That way, FeIDo can securely enable eID denylisting without requiring a trusted third party but client-side certificate storage and secure state synchronization of the eID denylist by the credential services.

Third, web services can block users on a per-eID basis. The credential service provides web services with service-specific, pseudonymized eID identifiers id_{srv} that the web services can store together with a user's FIDO2 public key. That way, web services can deny specific eIDs to enable users to revoke old eIDs when logging in with new ones—all without requiring additional state on the credential service or client-side, but at the cost of small web services changes. The credential services can dynamically derive unlinkable eID identifiers id_{srv} for each service based on the eID number, type, and issuer: $id_{srv} = HMAC(s_{kdf}, rp_{id}|eid\_num|eid\_type|state)$. This approach faces a potential attack window until a replacement eID has been acquired (cf. § 4.2) and registered on the web services. However, users can immediately close it if they own a second eID (e.g., eID and ePassport) or leverage one of the previous two approaches, i.e., report the eID as stolen or denylist it.

4.5.4 Anonymous Credentials in FIDO2. FeIDo's credential service enables a relying party to query pseudonymous meta user attributes, so-called anonymous credentials, on every authentication (cf. § 4.4). The relying party can query these meta attributes (e.g., "is adult") using WebAuthn extensions [41] defined by FeIDo. On an authentication request, the middleware asks the user which of the requested meta attributes they want to share with the given relying party and caches the decision for future requests. Only meta attributes explicitly permitted by the user are requested of the credential service, so users stay in full control over their data. The credential service calculates the meta attributes based on the verified personal user attributes as part of the regular authentication process and includes them (e.g., "is adult: true") inside the signed WebAuthn response data (*authenticator data* [41]).

It is crucial that the relying party can verify that the provided meta attributes were not spoofed but computed by some genuine

FeIDo authenticator based on verified user attributes. Therefore, the credential service binds the public key hash of the WebAuthn key pair used for signing the credential generation response, called the "attestation key pair" [41], to the SGX attestation as authenticated user data (cf. § 4.5.2, step 3). That way, the credential service enables a relying party to link every user credential to a genuine service TEE. Thus, the relying party transitively knows that all meta attributes associated with the credential have also been generated by the (*code-wise*) same credential service and are therefore trustworthy.

4.6 Deployment and Failover

FeIDo's design enables a cost-efficient, flexible, and scalable deployment on commodity hardware. However, FeIDo also faces failover challenges that it must address to assure credential access.

4.6.1 Component Deployment. The FeIDo authenticator is compatible with the FIDO2 standard for web authentication and can therefore be transparently used for existing relying parties (cf. § 6). FeIDo leverages off-the-shelf eIDs on the client-side and relies on commodity TEEs for its credential service. That way, FeIDo becomes easily deployable and does not demand additional user hardware as required by dedicated hardware or virtual FIDO2 authenticators.

FeIDo's credential service can be securely deployed on any remote service. We envision individuals or organizations volunteering to operate multiple credential service instances in a cloud, similar to software mirrors or Tor [24] nodes. That way, the credential services become widely accessible and can be shared by many users, so deployment costs become negligible. As the attribute-based credentials require only an *initial* state on the credential service (the secret and TLS server key), several service instances can be deployed in parallel without state synchronization—enabling wide availability and load balancing. Users can then freely choose any of the services for authentication as long as the service shares the KDF secret required for deriving the requested user credential. We discuss the credential service's secret bootstrap and failover in § 4.6.3 and the possibility of local enterprise deployments in § 7.2.

FeIDo's client middleware can be deployed as a regular application on any client device that supports an eID reader interface. The typical eID interfaces include NFC, USB, and BLE and are thus widely available (§ 2.2). If no reader is supported, e.g., on a workstation, the middleware can be offloaded to an auxiliary device, such as a smartphone (§ 4.5.2). The presented WebAuthn integration of the middleware via a browser extension is compatible with commodity browsers, as demonstrated by our prototype (§ 6.1).

4.6.2 Update Management. FeIDo's design as a virtual FIDO2 authenticator enables flexible update management. In contrast to dedicated hardware authenticators, all components of FeIDo can be updated via software or microcode updates. Therefore, the components can be quickly patched without extra (hardware) costs for users or credential service providers. FeIDo's attribute-based credentials enable users to seamlessly use any replacement eID issued by the same state that includes the same verifiable user attributes (§ 4.3) because the KDF will derive the same credential keys (*R3*).

One typical but infrequent corner case occurs if user attributes change, such as names after marriage. Such changes result in different WebAuthn credentials and thus require a user to re-register new credentials for their accounts. The same argument holds for users immigrating to other countries. Having said this, as FeIDo seamlessly accepts *any* valid, non-revoked user eID, users can use their "old" documents for migrating to newer attributes without risking account loss. We will discuss this approach further in § 7.3.

4.6.3 Secret Management. The credential service only requires minimal state on bootstrap: a KDF secret s_{kdf} and a TLS server key pair. The credential service can freshly generate the TLS server keys on each startup. If required for load balancing, the credential service can send a certificate signing request to a trusted certificate authority via TLS or even securely share the TLS server keys with other credential service instances via RA-TLS channels.

Secure management of s_{kdf} is crucial because s_{kdf} is part of the KDF input and binds the user credentials to the credential service instances (cf. § 4.3). A credential service can recover user credentials *only* if using the same s_{kdf} used for creation. Analogously, multiple credential service instances can derive the same user credentials *only* if they share the same s_{kdf}. Therefore, FeIDo and the credential service providers must ensure that s_{kdf} can be restored. At the same time, s_{kdf} must never be disclosed outside of the TEE protection domain to guarantee that it remains unknown to attackers.

We realize this guarantee as follows. The *initial* credential service instance generates a random s_{kdf}. The credential service then uses the Intel SGX sealing functionality which binds s_{kdf} to the credential service enclave and stores s_{kdf} encrypted and signed on the untrusted disk for recovery on a restart [19]. Sealing allows data recovery only on the same physical CPU. Should multiple credential service instances be scaled to different CPUs, they must cooperate to synchronize on the key. The credential service providers can configure new service instances to securely fetch s_{kdf} from other instances via mutually-attested RA-TLS channels. This way, s_{kdf} is *only* accessible by verified, TEE-protected credential services and is resilient against partial service failures or data loss.

5 SECURITY ANALYSIS

In this section, we give a security analysis of FeIDo (§ 5.1 and § 5.2), discuss the implications of different component thefts (§ 5.3), and conclude with an analysis of FeIDo's anonymous credentials (§ 5.4).

5.1 FeIDo's FIDO2 Security

We now provide arguments about the security of FeIDo and leave a more formal security proof of our scheme open to future work. We show that based on the security of the building blocks of FeIDo, it can be seen as a standard virtual authenticator that uses a keyed pseudorandom function to generate FIDO credentials. As shown by Hanzlik, Loss, and Wagner [29], such a virtual token is formally secure against a man-in-the-middle attacker residing between the agent and the relying party. The authors also prove the unlinkability of the generated FIDO credentials. We will consider two types of attackers: (A1) a man-in-the-middle attacker residing between the agent and the relying party, and (A2) an attacker that additionally controls the communication between the components of FeIDo.

Man-in-the-middle attackers (A1) cannot distinguish whether they interact with FeIDo or a standard virtual authenticator. It is in line with the design goals specified in § 3.1, where we state that FeIDo must comply with the FIDO2 protocols (*R1*). This also

means that we can directly apply the result from [29], assuming the function used to derive the credentials is pseudorandom. In our prototype, we use HMAC (cf. § 4.3.2, Equation 1), which Bellare showed to be a pseudorandom function [12]. Therefore, the security of FeIDo follows from [29], i.e., FeIDo provides unlinkable FIDO credentials and is secure against a man-in-the-middle attacker residing between the agent and the relying party.

5.1.1 Reduction Security of FeIDo: We now discuss the case where an attacker can additionally interact with the components of FeIDo (A2). We argue that assuming the credential service enclave is a secure TEE (§ 5.2.1), the KDF secret is not accessible outside valid credential services (§ 5.2.1), attributes used as input are unique (§ 5.2.3), and the eID is secure and the authority issuing the eID is trusted (§ 5.2.2), it is not easier to break the security of FeIDo than in the just discussed case where the attacker cannot interact with the components of FeIDo (A1). Trusting the issuing authority also assumes that an insider attacker cannot apply for an eID for different personal data, e.g., to launch a mimicry attack. In § 5.2, we show that those assumptions are reasonable.

Under those assumptions, the only difference between FeIDo and a virtual token is the communication between components. Therefore, we argue the security of FeIDo by reduction, starting from the secure Extended Access Control (EAC) protocol of ICAO-compliant eIDs—an authenticated key exchange protocol between an eID and a terminal. EAC has been proven to be secure by Dagdelen and Fischlin [20]. The protocol combines eID authentication using CA and PA (§ 2.2) with certificate-based authentication of the terminal, called *Terminal Authentication* (TA) and described in [47]. In EAC, an eID reader communicates with a terminal using TLS to proxy CA, PA, and TA for mutual eID and terminal authentication. We now show how FeIDo's components correspond to the ones used by the EAC security model in [20]. In both cases, the role of the eID remains the same. We simplify the function of the client middleware to that of an RFID/NFC reader. Finally, the credential service is the terminal verifying the eID. We will now briefly argue that the minor changes in FeIDo do not influence security. We begin by showing that RA-TLS provides the same security guarantees as a TLS connection and implies a trusted credential service. These two results allow us to argue the security of FeIDo without executing Terminal Authentication and directly apply the results from [20] since the other subprotocols used (CA and PA) are the same.

RA-TLS security: The RA-TLS protocol combines the guarantees of TLS with the remote attestation of a TEE (cf. § 4.5.2, step 3). Therefore, it provides an end-to-end protected connection between the client middleware and a verified credential service instance—protecting against network attackers and spoofed credential services. It follows that the RA-TLS connection satisfies the security properties of the TLS connection between the eID reader and terminal as required by the EAC model.

EAC security: The EAC protocol considers authentication of the terminal and the eID. The credential service is protected inside a TEE against system-level attackers at the service provider platform. In addition, the middleware remotely attests the protection and exact code of the credential service as part of the RA-TLS connection establishment. In combination with the credential service's open source design—allowing for code audits (cf. § 6.1)—users can validate the service's correctness and integrity. Under the assumption that the credential service executes in a secure TEE (§ 5.2.1), it follows that the credential service, i.e., the terminal, is trusted. It also means that the security of the whole EAC protocol holds even if, in FeIDo, we do not execute Terminal Authentication. Finally, the Passive Authentication protocol (PA, cf. § 2.2) provides the authenticity of the public key used by the eID during Chip Authentication (CA, cf. § 2.2), which is required by the EAC proof.

5.2 Security Assumption Verification

In the following, we discuss and verify our security assumptions made in FeIDo's security argument (cf. § 5.1.1).

5.2.1 Credential Service TEE and Secret Security. The TEE protection of the credential service is crucial for FeIDo to protect the KDF secret and thus the users' FIDO2 credentials against malicious service providers and spoofed services. Furthermore, remote attestation ensures that all communication parties can validate whether they communicate with a genuine credential service.

The TEE-assisted protection of the secret guarantees that an attacker who knows all user attributes cannot forge credentials. To derive the same secret key as the authenticator, the attacker would need to retrieve the credential service's secret key s_{kdf} for the KDF. The credential service protects s_{kdf} using the runtime protection and secure storage mechanisms of its TEE. The credential service shares s_{kdf} *only* with other verified credential services via mutually-attested, end-to-end protected connections (cf. § 4.6.3). Furthermore, relying parties can detect abuse of leaked secrets (cf. § 5.3.3).

5.2.2 eID Security and Malicious eID Authority. FeIDo relies on secure eIDs and a trusted eID authority to securely bind credentials to users based on their eIDs. FeIDo assumes the eIDs are free of backdoors and deploy typical smartcard protections, such as unclonable memory, constant-time cryptographic operations, and authentication mechanisms (see § 2.2, § 3.2). That way, FeIDo can rule out attackers trying to clone valid user eIDs for account hijacking. Thus, FeIDo follows similar assumptions as dedicated hardware authenticators, where trust is put into the authenticator devices and their manufacturing vendors. Note that for eIDs, cross-authority attacks are impossible, i.e., a malicious state trying to create valid eIDs of another. eIDs include the issuing country and are signed by a country-specific key checked during Passive Authentication. However, a malicious country M can still issue an eID that matches all attributes of an existing eID of a target country T *except* the issuing country code T. Therefore, FeIDo explicitly includes the issuing country as input to the KDF ($state$, Equation 1), resulting in *different* credentials. Thus, a malicious state (a.k.a. country) cannot issue eID clones to hijack accounts of residents of a different state.

5.2.3 Uniqueness of Personal Attributes. FeIDo's KDF uses a person's full name, date of birth, place of birth, and state as input (cf. § 4.3, Equation 1). If two persons of the same state are born on the same date, in the same city, *and* share the full name, they share the same credentials. The likelihood of such a collision highly depends on the city's respective naming convention and size. Without perfect global data to precisely measure this risk, we can only give approximations. For example, consider the US. Sweeney showed

that place, gender, and date of birth could uniquely identify half of the US population [51]. Surprisingly, this is *without* considering the name and surname, which introduce significant entropy. More generally, assume n users are namesakes born in the same state, city, and year. Then, assuming $d = 365$ days per year, based on the birthday paradox, $P = \frac{d-1}{d} * \frac{d-2}{d} * \ldots * \frac{d-(n-1)}{d}$ is the probability that none of these users share the same birthday. The resulting probabilities generally follow a Gaussian-like curve, but for smaller groups of namesakes, the chance for a collision decreases quadratically with the number of namesakes. To give upper bounds for the likelihood of a *full* collision in the US, we use the data from the US birth names inventory [4]. Consider the most common female (Olivia) and male name (Liam), combined with the most common surname (Smith). On average, even in the largest city of the census (NYC), there were just around $n = 8$ persons born with this name combination in 2019—resulting in a 92% chance even these US "worst-case combinations" of name/place/date are unique. Given the example of the US, one can argue that most users have unique personal data and hence KDF inputs. Admittedly, this argument may differ in naming conventions with highly-skewed name distributions. However, even in the unlikely event of full collisions, the security implications are limited in a 2FA setting. § 7.1 discusses how uniqueness can be extended for a single-factor setting.

5.3 FeIDo Component Theft (or Loss)

We now discuss how the theft (or loss) of the client device, eID, or KDF secret affects the security of FeIDo.

5.3.1 Client Device Theft. By design, FeIDo only runs the *secretless* client middleware on the client devices, which is not involved in the attribute-based credential derivation but only serves as a message proxy between the eID and credential service (§ 4.5). Therefore, theft of a client device has no security impact on FeIDo and, in contrast to client-side virtual FIDO2 tokens (cf. § 2.1.1 and § 4.2), does not stop a user from accessing their credentials.

5.3.2 eID Token Theft. At first glance, FeIDo seems to share the security guarantees of hardware tokens upon token/eID theft: Attackers could use the stolen eID for authentication operations. While in a 2FA setting—FeIDo's main use case—a stolen eID (or token) is *not* sufficient for attackers to hijack user accounts, attackers *could* gain access if they have *additionally* compromised the primary authentication factor or if FeIDo is used as a sole authenticator (§ 7.1). Similar to hardware tokens, the credential services could restrict service to eID types that support a PIN known only to the owner to prevent abuse by attackers. Then again, FeIDo enables for additional protections that off-the-shelf hardware tokens do not offer. In contrast to hardware tokens (§ 4.2), FeIDo *does support* central and per-service revocation of stolen eIDs, which blocks account hijacking attempts (§ 4.5.3). Furthermore, on an eID loss, users neither need backup tokens, less secure login alternatives, nor re-registration of new credentials. Instead, FeIDo's attribute-based credentials enable secure credential recovery (and thus account recovery) directly via replacement eIDs.

5.3.3 KDF Secret Theft. FeIDo's KDF secret s_{kdf} is crucial for deriving the user credentials. Therefore, the credential service uses its TEE to protect s_{kdf} against theft by restricting access to s_{kdf} only to secure, verified service instances (cf. § 5.2.1). Furthermore, FeIDo protects s_{kdf} against data loss and partial failures by combining TEE-protected backups with secure credential service replication (cf. § 4.3.2). That way, FeIDo can guarantee credential availability and avoid s_{kdf} changes that would require credential re-registrations. Note that even if the secret should ever leak, offline-calculated credentials will still not be accepted by the relying parties—remote attestation of the valid credential service can ensure that the credentials were genuinely derived (cf. § 4.5.4).

5.4 Security of Anonymous Credentials

We now discuss the security guarantees of FeIDo's anonymous credentials (§ 4.4) and their implications on an eID theft (§ 5.3.2).

5.4.1 Anonymity / Unlinkability Guarantees. FeIDo's anonymous credentials enable a relying party only to learn pseudonymous meta user attributes. The credential service only allows for meta attributes that provide a sufficiently large anonymity set such that the anonymous credentials provide reasonable group anonymity—even when used in conjunction. In addition, users have full control over the attribute sharing because they can allow/block any meta attribute on a per-relying party basis and must explicitly perform an authentication to allow for queries by the relying party (cf. § 4.4 and § 4.5.4). This rules out (mass) query attempts with the goal of user deanonymization. In the following, we assume the meta attributes to be anonymous and defer their full specification to future work.

Even though FeIDo's anonymous credentials are bound to the FIDO2 authentication process (§ 4.5.4), they do not enable linking attacks. While a relying party can notice via an SGX attestation report that the meta attributes are coming from some credential service, i.e., FeIDo authenticator, different relying parties can still not link multiple authentications to the same FeIDo instance. First, we showed that FeIDo is a FIDO2 authenticator (cf. § 5.1), i.e., its credentials and authentication operations are unlinkable. Second, the anonymous credential integration introduces *no* linkable information because (1) the meta attributes are anonymous, (2) SGX's attestation provides unlinkability [19], and (3) credential services use fresh per-credential attestation keys (cf. *AnonCA* [41]).

5.4.2 Impact of eID Theft. The anonymous credentials preserve FeIDo's security guarantees and easy account recovery on an eID theft (§ 5.3.2). However, while FeIDo's eID revocation prevents *any* abuse of stolen eIDs, *unnoticed* theft, which does not lead to an eID revocation, can be a real threat in practice for bypassing access policies that are based on a user's anonymous credentials. For instance, non-adult attackers might *temporarily* steal an eID from an adult (e.g., a parent) to bypass age gates. To prevent such attacks, FeIDo's credential service could restrict service *only* to eIDs that support an access PIN only known to the eID owner (§ 5.3.2).

6 EVALUATION

We now describe our FeIDo prototype and evaluate its performance.

6.1 Prototype

Our current prototype consists of an Android app, a browser extension, and an SGX enclave. For the sake of demonstration, without losing generality, we picked a German ePassport as eID—but

could have used any ICAO-standardized eID. Our prototype is open source[1] (see § 1) and currently focuses on the core concepts for web authentication. It does not yet support anonymous credentials.

6.1.1 Client Middleware. An Android app represents the client middleware. It provides a UI for manually entering the eID's static password as required for PACE—a one-time process that can be replaced by taking a photo of the user's eID (cf. § 2.2). The client middleware registers an NFC intent filter for detecting eIDs and uses the JMRTD library [42] for NFC-based communication with them, e.g., for PACE and data group queries. The client middleware communicates via protobuf messages with the credential service and browser extension through an RA-TLS (credential service) and WebSocket connection (browser extension), respectively. The client middleware enables the CA channel between the eID and the credential service by forwarding raw Application Protocol Data Unit [21] commands. As a performance optimization, the client middleware can cache data groups read from the user's eID to avoid re-reading them via NFC. Having said this, PACE, PA, and CA are re-executed on every authenticator operation to verify the eID (§ 4.3.1, § 4.5.2).

6.1.2 Browser Extension. The browser extension uses a content script to overwrite the create() and get() functions of a browser's navigator.credentials API. That way, the browser extension intercepts WebAuthn requests by a web page and can forward them to the FeIDo authenticator through the client middleware. This approach enables easy integration of FeIDo without emulating a physical device (e.g., USB). The browser extension manually crafts respective return values for the overwritten functions based on the client middleware's WebAuthn response data to interact with the web pages seemingly. While we tested the browser extension with Firefox, the core APIs are also available in other browsers.

6.1.3 Credential Service. The credential service is implemented based on the Intel SGX SDK v2.15 [17] in ~2.8 k lines of C/C++ code (excl. libraries). Upon an incoming RA-TLS connection by a client middleware, the credential service sequentially processes the WebAuthn operation request and eID information before replying with a WebAuthn response. We have integrated an RA-TLS server endpoint based on the SGX SSL patches of the SENG-SDK [50]. For parsing the eID data groups and running the PA and CA protocols, we patched the OpenPACE library v1.1.2 [26] to add support for German ePassports, SGX, and RA-TLS. To demonstrate eID revocation (§ 4.5.3), we have implemented a service that mimics Interpol's I-Checkit database service of stolen eIDs [30]. After CA, the credential service queries the blocklist via TLS using the eID's document number, type, and country of issuance (following [30]) and aborts authentication on a database hit. For the attribute-based FIDO2 credential derivation, the credential service uses a SHA-256 HMAC to hash the attributes and generates a WebAuthn public key credential based on the hash. The credential service prototype currently only supports ES256 for WebAuthn and German ePassports.

6.2 Performance Evaluation

We now evaluate the performance of our prototype implementation during a FIDO2 web authentication process.

6.2.1 Methodology. In our experiment, we host a local instance of the *webauthn.io* test page [36] and measure the time it takes to (i) register and (ii) log into an account using single factor FIDO2 authentication. We run the web service and the credential service on a Dell XPS 9560 laptop with an Intel® i7-7700 HQ and Ubuntu 18.04 LTS. For the agent, we use two devices: we run Firefox 96 and FeIDo's browser extension on the Dell XPS but run the middleware on a Pixel 4a phone with Android 12. The laptop and phone are interconnected via a local 1 Gbps network using Ethernet and Wifi.

To assess the practical feasibility of FeIDo, we compare its performance against two FIDO2 authenticators. For our measurements, we take a SoloKey Hacker v2.1 with an unlocked 4.1.2 firmware and a Nitrokey with a 2.4.0 firmware, which are connected via USB to the laptop, as baselines. We measure the average time over ten iterations for each operation (registration, login) for each authenticator. We keep the WebAuthn attestation feature of *webauthn.io* disabled for the measurements and assume the web page and client middleware app (with cached PACE password) to be preloaded. We assume that the ePassport is already placed on the NFC reader.

6.2.2 Evaluation Results. There were no significant time differences between the registration and login operations for all authenticators. We have measured the performance of our FeIDo prototype two times: once the initial, uncached performance (*uncached*) and once with the data groups of the ePassport cached by the client middleware (*cached*) as described in § 6.1. The uncached FeIDo operations took ~2980 ms on average, which is close to the average performance of Nitrokey of ~3183 ms. The overall median duration of Nitrokey has been ~2327 ms and the average duration of SoloKey ~1813 ms. The standard error of the means (SEM) of both FeIDo setups was below 28 ms and those of SoloKey below 101 ms. Nitrokey faced bigger SEMs of ~669 ms (register) and ~844 ms (login) due to outliers caused by its unreliable user confirmation based on squeezing the authenticator case in contrast to SoloKey's button.

By caching the ePassport data groups in the client middleware, FeIDo has significantly improved its average operation duration by ~19.3 % to ~1878 ms, which is close to that of SoloKey. This shows that reading ePassport data via PACE contributes a major part. In fact, our measurements have shown that the overall communication processing between the client middleware and ePassport currently makes up ~68 % to 78 % of FeIDo's operation duration. The total operation duration could probably be further improved by parallelizing the prototype, e.g., by setting up the PACE and RA-TLS connections concurrently or deriving the WebAuthn credentials while waiting for the PA and CA results. The current measurement excludes the eID revocation lookup as we have no access to the Interpol I-Checkit service. However, [30] states a lookup time of ~30 ms plus network latency, which is an insignificant extra overhead. We conclude that the performance of our current FeIDo prototype is already in the range of existing hardware FIDO2 authenticators.

7 DISCUSSION

In the following, we discuss FeIDo's applicability to single-factor (§ 7.1) and enterprise authentication schemes (§ 7.2) and how FeIDo operates in settings where personal user attributes change (§ 7.3).

7.1 FeIDo as Sole Authenticator (Passwordless)

While we assume FeIDo to be used as an additional factor in a 2FA scheme, in the following, we want to discuss in how far FeIDo could serve as a *sole* authenticator for FIDO2 web authentication. As FeIDo is a FIDO2-compliant virtual authenticator (cf. § 5.1), it can be directly used as the sole token. However, to provide high security in a non-2FA setting, FeIDo requires additional protection.

On an eID theft, just as hardware tokens, FeIDo must prevent account hijacking by attackers. To handle this, as proposed in § 5.3.2, FeIDo's credential services can restrict service to eIDs that support an access PIN known only to the genuine owner—analogous to PINs provided by some hardware tokens—and implement a suitable subset of FeIDo's eID revocation mechanisms (cf. § 4.5.3). That way, stolen eIDs become unusable by attackers not knowing the PIN or even entirely revoked for FeIDo authentication operations.

While a collision of our chosen set of user attributes is unlikely in practice (§ 5.2.3), when considering FeIDo for usage as the *sole* authenticator, we must *guarantee* user-unique KDF inputs to derive distinct credentials. To this end, we could add a user-specific secret *salt* to the KDF input set (§ 4.3.2). This secret must be a strong, client-side random password accessible by the client middleware, which is forwarded to a credential service as part of the authentication request. That way, even on a full attribute collision of two users, their KDF input and thus derived credentials stay distinct. While the secret can be cached on the client device, e.g., in a phone's TEE-protected key storage [6], the user has to back up the salt against client device loss. As a positive side effect, the salt also protects against eID theft and cloned eIDs issued by malicious authorities.

The depicted sole-factor usage of FeIDo allows to draw a comparison to sole-factor passwords for web authentication. All FIDO2 schemes, including FeIDo, are resilient against several attacks that passwords do not withstand, such as phishing, shoulder surfing, password database leaks, or cracking/guessing attacks [22, 25, 45, 49]. Moreover, whereas password leaks give attackers immediate access to the associated account(s), the FeIDo/FIDO2 credentials cannot be extracted to impersonate users even if the attacker has physical access to the token (or eID). While attackers can *reuse* stolen/lost tokens to impersonate users, access codes (or the PIN/salt depicted above) mitigate this threat. Admittedly, passwords have lower requirements: they are widely supported, do not have to be carried, and require no special or dedicated hardware. For a more detailed comparison between sole-factor FIDO2 and passwords, we refer to Lyastani et al.'s systematization paper [27].

7.2 Enterprise Authentication Use Cases

So far, we have focused on FeIDo being used by private users. We now discuss FeIDo's applicability to enterprise-focused authentication use cases, loosely following those given in [10] by the FIDO Alliance. As FeIDo is fully FIDO2-compliant, in principle, FeIDo can be used for any enterprise setting where FIDO2 hardware or virtual authenticators are in use. Employees can directly use their eIDs with FeIDo for enterprise two- or single-factor web authentication. Alternatively, if a company issues electronic employee ID badges that provide the required personal attributes and compatible authentication protocols (§ 2.2), FeIDo could support them.

FeIDo can also be used for *local* enterprise service or device authentication if network connectivity is available. FeIDo requires access to a credential service and an attestation service for validating the TEE protecting the credential service. If internet connectivity is available (default), both services can be hosted remotely, e.g., in a public cloud and by the TEE vendor. Alternatively, to enable *local* intranet settings, a company can host a private credential service and attestation service (e.g., Intel SGX DCAP [18]) on a local enterprise server. That way, FeIDo can even be used for other settings such as local client device (domain) logins, remote logins, or SSH logins. FeIDo can even support physical access, e.g., via smart door locks, as smart locks often support NFC and intranet access for checking credentials against the enterprise database.

FeIDo's requirements are thus comparable to those of existing FIDO2 authenticators, except authenticators do not require access to a TEE-protected credential service, which simplifies local setups. In addition, they can better support offline use cases.

7.3 eID Migration on Attribute Changes

FeIDo's KDF-based credentials rely on the fact that user attributes do not change. While this is true for most attributes such as date and place of birth, names or nationality *may* change. For example, users may change their surname upon marriage. Since the credential derivation is based on a user's attributes, the user loses access to the relying parties once attributes change. This migration problem can be solved by temporarily switching to other authentication schemes and eventually (re-)linking the (new) user data to the account. Alternatively, users can leverage the fact that they may own multiple valid eIDs issued by the same state (e.g., a national ID *and* an ePassport). FeIDo transparently accepts *any* valid, non-revoked eID of a user. The user can first apply for just one of the two documents with their new data. They can then still use the "old" document for a final authentication based on their outdated data and then link their new document to their accounts. Once all accounts are migrated, the user can finally replace the other eID.

8 CONCLUSION

FeIDo represents the first attribute-based FIDO2 virtual authenticator. The system uses an eID as the physical component that the user possesses and a protected credential service that interfaces the eID's authentication mechanism with the FIDO2 protocol. FeIDo addresses two major open challenges in FIDO2: cost efficiency, and recovery in case of authenticator loss. We also show that it is secure under reasonable assumptions. In contrast to existing FIDO2 tokens and eIDs, FeIDo additionally enables anonymous credentials, which the credential service can provide as a FIDO2-compatible extension.

Finally, we provide an open-source prototype of FeIDo and compare its efficiency with existing hardware authenticators. In particular, we show that the execution time of our prototype implementation is comparable with standard FIDO2 authenticators. We conclude that FeIDo is a practical, cost-efficient, and secure alternative to existing hardware and virtual authenticators.

REFERENCES

[1] 2019. *Regulation (EU) 2019/1157 of the European Parliament and of the Council.* https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32019R1157

[2] 2020. *Standardized Digital Identity on National Identity Cards.* https://www.calctopia.com/2020/02/14/standardized-digital-identity-on-national-identity-cards/

[3] 2021. *National ID cards: 2016-2021 facts and trends.* https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/identity/2016-national-id-card-trends

[4] 2021. *Popular Baby Names (US).* https://www.ssa.gov/oact/babynames/limits.html

[5] 2021. *The electronic passport in 2021 and beyond.* https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/passport/electronic-passport-trends

[6] 2022. *Hardware-backed Keystore.* https://source.android.com/security/keystore

[7] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. 2005. Password-Based Authenticated Key Exchange in the Three-Party Setting. In *Public Key Cryptography*, Serge Vaudenay (Ed.). Springer Berlin Heidelberg, 65–84.

[8] FIDO Alliance. 2020. *Using FIDO with eIDAS Services.* https://fidoalliance.org/wp-content/uploads/2020/04/FIDO-deploying-FIDO2-eIDAS-QTSPs-eID-schemes-white-paper.pdf

[9] FIDO Alliance. 2021. *FIDO Security Reference.* https://fidoalliance.org/specs/common-specs/fido-security-ref-v2.1-rd-20210525.html

[10] FIDO Alliance. 2022. *Choosing FIDO Authenticators for Enterprise Use Cases.* Retrieved July 28, 2022 from https://media.fidoalliance.org/wp-content/uploads/2022/03/FIDO-White-Paper-Choosing-FIDO-Authenticators-for-Enterprise-Use-Cases-RD10-2022.03.01.pdf

[11] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2021. CURE: A Security Architecture with CUstomizable and Resilient Enclaves. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1073–1090. https://www.usenix.org/conference/usenixsecurity21/presentation/bahmani

[12] Mihir Bellare. 2015. New proofs for NMAC and HMAC: Security without collision resistance. *Journal of Cryptology* 28, 4 (2015), 844–878.

[13] Jens Bender, Marc Fischlin, and Dennis Kügler. 2009. Security Analysis of the PACE Key-Agreement Protocol. In *Information Security*. Springer Berlin Heidelberg, 33–48.

[14] Inc. Biometrics Research Group. 2020. *Apple launches web authentication using FIDO standard with Touch ID or Face ID biometrics in Safari.* https://www.biometricupdate.com/202006/apple-launches-web-authentication-using-fido-standard-with-touch-id-or-face-id-biometrics-in-safari

[15] Dhiman Chakraborty and Sven Bugiel. 2019. SimFIDO: FIDO2 User Authentication with simTPM. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2569–2571.

[16] Dhiman Chakraborty, Lucjan Hanzlik, and Sven Bugiel. 2019. simTPM: User-centric TPM for Mobile Devices. In *28th USENIX Security Symposium (USENIX Security 19)*. 533–550.

[17] Intel Corporation. [n. d.]. *Intel SGX for Linux.* https://github.com/intel/linux-sgx

[18] Intel Corporation. 2022. *Intel SGX Data Center Attestation Primitives.* https://download.01.org/intel-sgx/sgx-dcap/1.14/linux/docs/DCAP_ECDSA_Orientation.pdf

[19] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016).

[20] Özgür Dagdelen and Marc Fischlin. 2011. Security Analysis of the Extended Access Control Protocol for Machine Readable Travel Documents. In *Information Security*. Springer Berlin Heidelberg, Berlin, Heidelberg, 54–68.

[21] Organización Internacional de Normalización. 2020. *ISO IEC 7816-4: Identification cards–Integrated circuit cards. Organization, security and commands for interchange.* ISO.

[22] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. 2010. Password strength: An empirical analysis. In *2010 Proceedings IEEE INFOCOM*. IEEE, 1–9.

[23] Ghada Dessouky, Tommaso Frassetto, and Ahmad-Reza Sadeghi. 2020. HybCache: Hybrid Side-Channel-Resilient Caches for Trusted Execution Environments. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity20/presentation/dessouky

[24] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium (USENIX Security 04)*. USENIX Association, San Diego, CA. https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router

[25] Malin Eiband, Mohamed Khamis, Emanuel Von Zezschwitz, Heinrich Hussmann, and Florian Alt. 2017. Understanding shoulder surfing in the wild: Stories from users and observers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 4254–4265.

[26] Frank Morgner and Dominik Oepen. [n. d.]. *OpenPACE.* https://frankmorgner.github.io/openpace/

[27] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. 2020. Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication. In *IEEE Symposium on Security and Privacy (SP)*.

[28] Sérgio Gonçalves, Alessandro Tomasi, Andrea Bisegna, Giulio Pellizzari, and Silvio Ranise. 2020. *Verifiable Contracting: A Use Case for Onboarding and Contract Offering in Financial Services with eIDAS and Verifiable Credentials.* 133–144. https://doi.org/10.1007/978-3-030-66504-3_8

[29] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. 2022. *Token meets Wallet: Formalizing Privacy and Revocation for FIDO2.* https://ia.cr/2022/084

[30] Interpol. 2022. *I-Checkit - FAQs brochure - Private Sector Partners.* Retrieved July 25, 2022 from https://www.interpol.int/content/download/12470/file/I-Checkit_FAQs_brochure_private%20sector_EN_LR_02.pdf?inLanguage=eng-GB

[31] Janis Danisevskis. 2018. *Android Protected Confirmation: Taking transaction security to the next level.* https://android-developers.googleblog.com/2018/10/android-protected-confirmation.html

[32] Governikus GmbH & Co. KG. 2022. *AusweisApp2: Passende Smartphones & Tablets für die Online-Ausweisfunktion.* Retrieved July 27, 2022 from https://www.ausweisapp.bund.de/mobile-geraete

[33] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. 2018. Integrating Remote Attestation with Transport Layer Security. *CoRR* abs/1801.05863 (2018). arXiv:1801.05863 http://arxiv.org/abs/1801.05863

[34] Hugo Krawczyk. 2010. Cryptographic extraction and key derivation: The HKDF scheme. In *Annual Cryptology Conference*. Springer, 631–648.

[35] Johannes Kunke, Stephan Wiefling, Markus Ullmann, and Luigi Lo Iacono. 2021. Evaluation of Account Recovery Strategies with FIDO2-based Passwordless Authentication. In *Open Identity Summit*. Gesellschaft für Informatik e.V., Bonn.

[36] Duo Labs. 2020. *WebAuthn.io (Github).* https://github.com/duo-labs/webauthn.io

[37] Duo Labs. 2021. *WebAuthn.io: A demo of the WebAuthn specification.* https://webauthn.io/

[38] Zeyu Lei, Yuhong Nan, Yanick Fratantonio, and Antonio Bianchi. 2021. On the Insecurity of SMS One-Time Password Messages against Local Attackers in Modern Mobile Devices. In *28th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/on-the-insecurity-of-sms-one-time-password-messages-against-local-attackers-in-modern-mobile-devices/

[39] Blue Bite LLC. 2021. *Android NFC Compatibility.* Retrieved July 27, 2022 from https://www.bluebite.com/nfc/android-nfc-compatibility

[40] SJB Research Ltd. 2019. *Confirmed: iOS 13 to include support for NFC passport reading - NFCW.* Retrieved July 27, 2022 from https://www.nfcw.com/2019/06/07/362943/confirmed-ios-13-to-include-support-for-nfc-passport-reading/

[41] Emil Lundberg, Michael Jones, J.C. Jones, Akshay Kumar, and Jeff Hodges. 2021. *Web Authentication: An API for accessing Public Key Credentials - Level 2.* Technical Report. W3C. https://www.w3.org/TR/2021/REC-webauthn-2-20210408/

[42] Martijn Oostdijk. [n. d.]. *JMRTD: An Open Source Java Implementation of Machine Readable Travel Documents.* https://jmrtd.org/

[43] Frank Morgner, Paul Bastian, and Marc Fischlin. 2016. Securing Transactions with the eIDAS Protocols. In *Information Security Theory and Practice*, Sara Foresti and Javier Lopez (Eds.). Springer International Publishing, Cham, 3–18.

[44] Jämes Ménétrey, Christian Göttel, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. 2022. An Exploratory Study of Attestation Mechanisms for Trusted Execution Environments. In *Workshop on System Software for Trusted Execution*.

[45] Adam Oest, Penghui Zhang, Brad Wardman, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, Adam Doupé, and Gail-Joon Ahn. 2020. Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale. In *29th USENIX Security Symposium (USENIX Security 20)*. 361–377.

[46] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. 2018. Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks. In *USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, 227–240. https://www.usenix.org/conference/atc18/presentation/oleksenko

[47] International Civil Avaiation Organization. 2021. *Machine Readable Travel Documents Part 11: Security Mechanisms for MRTDs* (eighth ed.). Technical Report. https://www.icao.int/publications/documents/9303_p11_cons_en.pdf

[48] International Civil Avaiation Organization. 2021. *Machine Readable Travel Documents Part 3: Specifications Common to all MRTDs* (eighth ed.). Technical Report. https://www.icao.int/publications/Documents/9303_p3_cons_en.pdf

[49] Hamza Saleem and Muhammad Naveed. 2020. SoK: Anatomy of Data Breaches. *Proc. Priv. Enhancing Technol.* 2020, 4 (2020), 153–174.

[50] Fabian Schwarz and Christian Rossow. 2020. SENG, the SGX-Enforcing Network Gateway: Authorizing Communication from Shielded Clients. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 753–770. https://www.usenix.org/conference/usenixsecurity20/presentation/schwarz

[51] Latanya Sweeney. 2000. Simple demographics often identify people uniquely. *Health (San Francisco)* 671, 2000 (2000), 1–34.

[52] Yubico. 2021. *Losing Your YubiKey - Yubico.* https://support.yubico.com/hc/en-us/articles/360013647620-Losing-Your-YubiKey

[53] Yubico. 2022. *Spare YubiKeys.* https://www.yubico.com/spare/

[54] Yubico. 2022. *WebAuthn - Account Recovery.* https://developers.yubico.com/WebAuthn/WebAuthn_Developer_Guide/Account_Recovery.html