# Securing FIDO2 Credentials Using Intel SGX

Tomás Matos
*IEETA / LASI*
*University of Aveiro*
Aveiro, Portugal
0009-0008-4203-5849
tomas.matos@ua.pt

André Zúquete
*IEETA / LASI / DETI*
*University of Aveiro*
Aveiro, Portugal
0000-0002-9745-4361
andre.zuquete@ua.pt

Tomás Oliveira e Silva
*IEETA / LASI / DETI*
*University of Aveiro*
Aveiro, Portugal
0000-0002-8878-3219
tos@ua.pt

***Abstract*—As the digital infrastructure expands, new authentication methods have become popular in both enterprises and the research community, innovating and increasing the security of authentication processes. FIDO2 is a standard that aims to replace passwords using asymmetric cryptography through challenge-response mechanisms, keeping private keys stored in secure tokens. As with all technologies, it comes with drawbacks, such as the cost of physical devices and recovery mechanisms that make adoption harder. Due to these problems, this work proposes a centralized credential manager that stores these types of credentials and provides phishing-resilient authentication without requiring hardware investment and with replication mechanisms for fault tolerance. To tackle data breaches in the centralized application, we leverage Intel SGX capabilities, which provide a trusted execution environment and attestation mechanisms to ensure that users get the expected behavior when interacting with their credentials.**

***Keywords*—FIDO2, SGX, Phishing, Data Breaches, Credential Managers.**

## I. Introduction

Authentication through credentials has become fundamental to our digital lives, but it is facing serious challenges. Recent reports show worrying trends in how credentials are compromised, making it clear that we need better ways to authenticate users.

Phishing attacks are getting worse at an alarming rate. The KnowBe4 Phishing Threat Trends Report (2024-2025)[1] and the Cybercrime Information Center's 2025 Phishing Landscape[2] show that phishing attacks jumped by 17.3% in just six months. What is particularly concerning is that 82.6% of these malicious messages now use AI-generated content to slip past traditional filters. Check Point's analysis[3] adds to this picture: compromised credentials surged by 160% in early 2025, with over 14,000 employee credentials exposed in just one month. Attackers are getting better at exploiting human behavior through increasingly sophisticated AI-powered social engineering tactics.

The problems with traditional passwords are well documented. Research by the National Institute of Standards and Technology (NIST) highlights that users struggle to maintain unique passwords across multiple services [1], leading to widespread password reuse. When one service experiences a data breach, attackers leverage these credentials in credential stuffing attacks against other platforms. Verizon's Data Breach Investigations Report consistently shows that stolen credentials remain among the top attack vectors[4]. Even users who create strong, unique passwords for each service are not safe from phishing attacks that simply trick them into handing over their credentials.

Several solutions exist, but each comes with its own problems. Multi-Factor Authentication (MFA) adds an extra security step, making phishing much harder. But being honest: most users find it annoying to pull out their phone or wait for a code every time they want to log in. Studies on usability of authentication [2], [3] show that friction in authentication processes leads to reduced adoption rates. This friction leads many to avoid MFA altogether [4].

FIDO2 authentication[5] is genuinely phishing-resistant because it uses cryptographic keys instead of passwords that people can be fooled into revealing. The catch? You need to buy a hardware security key, carry it with you, and worry about what happens if you lose it.

Password managers offer a practical middle ground. They store all your passwords in an encrypted vault, generate random passwords for each service, and only require you to remember one master password. Many include MFA to access the vault itself. This nicely solves the password reuse problem. But there is a fundamental trust issue: you are putting all your eggs in one basket and hoping the password manager provider has perfect security. Recent incidents involving password manager breaches demonstrate this risk[6]. If their system gets compromised, or if they are not as trustworthy as they claim, all your credentials could be exposed at once. FIDO2 also evolved to being supported by password managers, such as

[1]KnowBe4 Research, *Phishing Threat Trends Report 2024-2025*, available at: https://www.knowbe4.com/phishing-security-test-report

[2]Cybercrime Information Center, *2025 Phishing Landscape Analysis*, available at: https://cybercrimeinfocenter.org/phishing-landscape-2025

[3]Check Point Research, *Cyber Security Report 2025: Compromised Credentials Analysis*, available at: https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-phishing/

[4]Verizon, *2024 Data Breach Investigations Report*, available at: https://www.verizon.com/business/resources/reports/dbir/

[5]FIDO Alliance, *FIDO2: WebAuthn & CTAP Specifications Overview*, available at: https://fidoalliance.org/fido2/

[6]Goodin, D., *Password Manager LastPass Warns of Breach Allowing Access to Customer Data*, Ars Technica, 2022, available at: https://arstechnica.com/information-technology/2022/12/lastpass-says-hackers-have-obtained-vault-data-and-a-wealth-of-customer-info/

Windows Hello[7], and currently the term passkeys refers to FIDO2 credentials that are managed by password managers instead of hardware tokens.

This paper explores how we can address these challenges using Intel SGX. The goal is to combine the convenience of centralized password managers with something traditional solutions lack: verifiable proof that the system is actually doing what it claims. By integrating SGX's attestation capabilities [5] with modern FIDO2 authentication, we can build a credential management system that does not require users to simply trust that the provider is being honest and secure.

The structure of the paper is the following. Section II summarizes the basic concepts of FIDO2 and SGX. Section III presents the solution that we propose Section IV presents the implementation of our solution in Linux systems. Section V analyzes the vulnerabilities identified in our solution. Section VI compares our solution with other proposed alternatives. Section VII concludes the paper and highlights some future work.

## II. Contextual overview

In this section, we will briefly review FIDO2 and SGX, in order to introduce the basic concepts that are fundamental to understanding our contribution.

FIDO2 uses asymmetric key pairs managed by an Authenticator, which is a personal device, to authenticate people in a Relying Party (RP), which is a service that needs to authenticate people. On a user registration, the RP stores the user's public key on their account; on the user's authentication, the RP requires a signature with the user's private key on a challenge (an assertion). FIDO2 allows users' Authenticators, which are the ones that compute those signatures, to keep the private keys, or secret material to generate them, inside them (resident credentials) or in the RP itself, as part of the user's account (non-resident credentials). Resident keys are discoverable, the token owner can list them. Our system uses only this strategy to allow backups (since you cannot backup something that you cannot list).

FIDO2 is based on two high-level protocols: WebAuthn [6] and Client to Authenticator Protocol (CTAP) [7]. WebAuthn describes how a web application running in a browser interacts with an RP to register and authenticate a user. CTAP describes how clients (e.g., browsers) discover and interact with local FIDO2 devices through several communication technologies (USB, Bluetooth, or NFC). Browsers provide run-time support that enables local Web applications to use CTAP transparently to obtain registration public keys or to produce a signature on a challenge. Our solution, illustrated in Figure 1, uses an Authenticator Emulator application that has a CTAP interface and uses local or remote Intel SGX enclaves to securely manage FIDO2 key pairs.

FIDO2 key pairs are identified by a Credential_ID defined by the device that created them. This identifier is what allows private keys to be recreated (when non-resident) or fetched (when resident) inside the token that produced them. A Credential_ID is intrinsically bound to an RP, which means that it is produced by a token for a given RP and can only be used to authenticate a user in the same RP. The domain name of the RP is a parameter transmitted by browsers to tokens on registration or authentication operations.

FIDO2 tokens have operational attributes that are sent to clients and RPs to assert the required collaboration of a token owner in registration and authentication actions. These include User Presence (UP, the user must do something to approve the action, such as pressing a token button), and User Verification (UV, the user needs to prove the token ownership, usually with a PIN or with a biometric recognition). In our system, UP and UV are implemented by the Authentication Emulator.

FIDO2 tokens offer excellent phishing protection [8], but they come with practical limitations. Users must purchase them, keep track of them, and have backup strategies in case they are lost or damaged [9], [10]. Beyond usability concerns, the CTAP protocol between clients, like browsers, and FIDO2 authenticators can also introduce vulnerabilities, particularly the Man-in-the-Middle (MITM) attack that exploits weaknesses in the Diffie-Hellman key exchange governing PIN negotiation, as well as the ability to tamper with critical messages, including authenticator metadata (fundamental to guide clients' behavior and security requisites) and the registration and authentication process themselves.

Our solution requires participation in this protocol to interact with all clients, not just browsers, meaning that the security risks that already exist remain a concern. We cannot guaranty that messages exchanged between these two entities are free from tampering by an intermediary. However, we consider a MITM attack on this protocol highly unlikely in practice, as it presupposes a local host compromise [11]. Some research efforts address these issues [12], but solving them falls outside the scope of this work. Finally, we mitigate some exposure by blocking and controlling certain dangerous CTAP commands, such as credential resetting instructions, and making our emulated authenticator resilient to the permission vulnerabilities described in [13].

Intel SGX deploys enclaves as a Trusted Execution Environment (TEE). Computations within enclaves cannot be observed by anyone, they run protected by the CPU. Enclave programs have an inbound API (set of e-calls) and an outbound API (set of o-calls). This is the only way they can communicate with the outside world. An enclave application is launched by an ordinary application that can call its e-calls and must implement its o-calls. Our SGX-based credential manager is made up of an application that has a Web interface and launches a specific enclave application to deal with FIDO2 cryptographic keys (see Figure 1).

An application running inside an enclave has an identity derived from its code (`MRENCLAVE`) and from the public key of its producer (`MRSIGNER`). Changing the enclave's code changes `MRENCLAVE`, thus it can be used to its attestation. `MRSIGNER` is mainly used to allow one enclave to recognize

[7]Microsoft, *Support for passkeys in Windows*, available at https://learn.microsoft.com/en-us/windows/security/identity-protection/passkeys/

other enclaves produced or deployed by the same organization. These values are produced by the CPU when a signed enclave application is loaded into an enclave.

An enclave can produce internal keys (seal keys) that can depend on `MRENCLAVE` or `MRSIGNER`. These keys can be used to encrypt exported data, which allows data to be stored outside the enclave without exposing its contents to others (enclaves are volatile environments with no persistent storage, enclave applications have to deal with this issue). These keys are CPU-dependent: different CPUs produce different keys for the same input parameters. When `MRENCLAVE` and the root sealing key are used, no other enclave application can reproduce those sealing keys; we used this approach. In some scenarios, such reproduction is desirable (e.g., to patch enclave applications without loosing sealed data), but that requires trusting on `MRSIGNER`, and we do not intend to do so.

Remote attestation of an enclave [5] is a process that allows anyone to check remotely if it is interacting with a running enclave with a given `MRENCLAVE` or `MRSIGNER`. For that, an enclave first produces a `REPORT`, which is a data structure that includes those two identifiers, plus some additional arbitrary data (e.g., a public key for securely communicating with the enclave) and a signature created with a Report Key. The Report Key is a key belonging to another local enclave that enables it to validate `REPORT` as something genuinely produced by the same CPU. There is a special Intel enclave called the Quoting Enclave, which produces a `QUOTE` from a `REPORT`. A `QUOTE` is fundamentally a `REPORT` signed with a certified key pair. This signature can be validated anywhere using the Intel X.509 PKI (Public Key Infrastructure) that was created for SGX remote attestation. Thus, a client application can remotely attest an enclave by getting its `QUOTE`, validating its signature using the PKI, and checking the `MRENCLAVE` value on it. Then, it can use the arbitrary data placed by the enclave in the initial `REPORT` to establish a secure connection with the attested enclave.

## III. Proposed Solution

This work tackles the several interconnected problems mentioned in the introduction on modern authentication systems. First, centralized credential managers, whether cloud-based or self-hosted, require users to trust that the service provider is secure and honest. By leveraging Intel SGX enclaves, this trust assumption is replaced by a verifiable guaranty: credentials are encrypted at rest and processed exclusively within a hardware-isolated execution environment, whose integrity can be remotely attested by any party.

Furthermore, considering the impact of data breaches on credential managers, our proposed storage model will not include identifiable material. This means that data stored by credential managers cannot be linked to FIDO2 private keys without the help of the code that runs inside an SGX enclave. Furthermore, we allow FIDO2 private keys to migrate among equal SGX enclaves, to enable redundancy of credential managers, but the stored data are always identified differently.

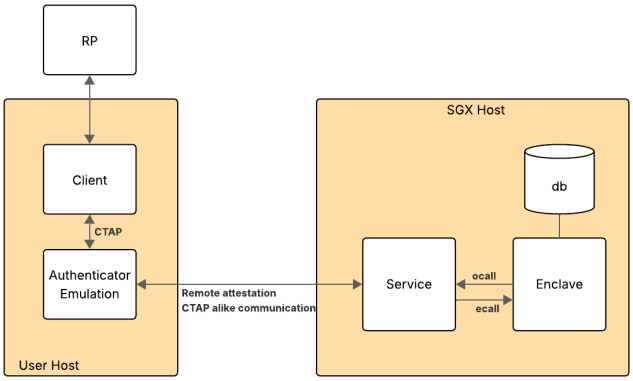


Fig. 1. Architecture of our SGX-based FIDO2 credential manager. The Client application would normally be a browser.

Figure 1 illustrates the proposed solution architecture, which consists of six main components: the WebAuthn-enabled RP, the Client (browser), the Authenticator Emulator, the SGX host service, the SGX Enclave and the SGX host storage, needed for storing resident keys.

### A. Authentication Emulator

The Authenticator Emulator functions as a software Authenticator, exposing a CTAP interface to clients, such as browsers. It handles directly most interactions with clients, except those that require the collaboration of an SGX host: initialization, authenticator information retrieval, and most importantly, credential registration and assertion [7]. The PIN related commands were not necessary to implement, as the authenticator declares the `uv` flag as `true` in the `authenticatorGetInfo` response, indicating that the authenticator is capable of built-in user verification. Although this typically implies biometric capabilities in conventional Authenticators, in our system, is performed through the cryptographic binding of the user's personal attributes. To improve usability, the user may select some attributes to be cached but not all (to effectively enforce user verification).

For managing user credentials, the Authenticator Emulator gathers user-provided attributes (personal information, passphrase, etc.) and generates a user identifier (UID) through an SHA-256 hash of those attributes. Each UID aggregates a set of FIDO2 credentials belonging to a user. It is possible for a user to manage different UIDs (e.g., by providing a different passphrase for each set of FIDO2 credentials, something similar to having different hardware tokens for different purposes).

Communication between the Emulator and the enclave, through the SGX host Service, is secured using an Elliptic Curve Integrated Encryption Scheme (ECIES) using an EC public value fetched from a valid `QUOTE` produced for the enclave. This prevents sensitive data, including the UID and RP identifiers, from being observed by others outside of the enclave.

### B. Key pair creation and recreation

Within the enclave, a hardware-bound Identity Key (IK) is deterministically generated on each run using the SGX seal key mechanism. This key is used to create key pairs and their Credential_ID using a deterministic derivation via HMAC and KDF operations from a UID, an RP and the user identifier in the relying party:

$$\begin{aligned} \text{key pair} &= \text{KDF}\left(\text{IK}, \text{Credential\_ID}\right) \\ \text{Credential\_ID} &= \text{HMAC}_{\text{IK}}\left(\text{UID}, \text{RP}, \text{user}\right) \end{aligned}$$

Since the key pair is derived from non-random data, no credential private material needs to be stored persistently, and there is nothing to breach at rest.

During authentication, the user presents the same attributes (UID and RP) and the Credential_ID provided by the RP. Note that the user identifier in the relying party is not sent along in the authentication request, meaning that to derive the key pair, we need to store a tuple containing:

$$\left(\text{UID}, \text{RP}, \text{user}, \text{Credential\_ID}\right)$$

In a file (`key-pair`) during the registration phase. For each authentication request, the enclave searches the sent Credential_ID, in the file, to recreate the exact same key pair and produce a valid assertion for the RP challenge.

This storage file is not only necessary for authentication. To support migration, the enclave must retain this information to know which credentials exist, either produced locally or imported from another enclave. With this file, it is possible to associate each UID with a list of RPs with which they have registered, so that all credentials can be derived and transferred. To identify which credentials were migrated, is necessary to look through a fifth attribute in the tuple corresponding to the migrated key pair.

This file is encrypted with IK; thus, only the enclave has access to its contents. If a breach occurs, an attacker gains access only to an encrypted file that they cannot decrypt. Since it is a single file, there is no external clue about which UID and RP motivated its update upon a new credential registration.

Tying the credential derivation to the SGX sealing key saves storage space on the SGX host that creates the credentials and binds them to intrinsic security features of SGX. But this hardware dependency implies that credentials imported from other enclaves need to have additional data to recreate them. Thus, the migration must be carefully designed in order to protect the migrated credentials with a security level similar to the one they had in their original creation environment.

### C. Credential migration

Credential migration among SGX hosts (or their enclaves) introduces additional complexity, since the processor changes, and so does IK, making direct key pair derivation impossible on the destination enclave. Therefore, the migration of the credentials of a given UID from one enclave where they were created to another implies either (i) sending the IK and the list of RPs and users belonging to that UID to the destination enclave, or (ii) sending the key pairs belonging to the same list of RPs and users. The first approach is dangerous, as it would expose IK outside of its creator enclave, therefore we decided to use the second one.

The migration process is initiated and mediated by the Authenticator Emulator, following some ideas presented in [14]. First, it gets the list of RPs from the current UID stored in the SGX host, to allow the user to select the ones they intend to migrate. Second, it gets a `QUOTE` from the destination enclave, validates it, and sends it to the current one, together with the list of RPs for which the key pair should be migrated. The enclave validates the `QUOTE`, and uses the public key on it (which belongs to the target enclave) to encrypt the key pairs to migrate. Finally, the Authenticator Emulator sends the UID, the list of RPs and the corresponding encrypted set of key pairs to the destination enclave. The enclave then adds the credential tuples to its `key-pair` with the key pair included. Naturally, the contents of this last file are also encrypted with the local IK.

Note that the SGX host with the destination enclave is not just a passive backup -— it is a fully functional instance capable of registering new credentials, authenticating users, and migrating credentials to other SGX hosts upon request. This means that two types of credential coexist on any given host: those derived natively from the current hardware, and those imported from another enclave and stored explicitly. Both types of credentials can be migrated to other SGX hosts following the same process.

As mentioned above, the destination enclave must validate the `QUOTE` received during migration, which requires verifying the embedded certificate chain. A practical challenge arose here: in the Emulator, the Intel SGX Provisioning Certification Root CA is fetched at runtime from Intel's servers[8]. However, inside the enclave, outbound network requests are not possible due to the security guaranties SGX enforces, and asking the SGX host for a root certificate means delegating trust on a critical element to an untrusty component. To work around this, the root CA certificate was inspected and, since it is valid until 2049, it was hard-coded directly into the enclave code, making chain verification possible without any external dependency.

This approach carries a long-term concern. When the certificate eventually expires, the enclave code must be updated, which changes the `MRENCLAVE` measurement. Any client that pins trust to the old measurement will need to be updated and made aware of the change, adding a maintenance burden that is necessary to acknowledge. Furthermore, credentials protected with keys derived from an old `MRENCLAVE` cannot be used by a new version; consequently, users would have to create new credentials for their RPs.

[8]https://certificates.trustedservices.intel.com/Intel_SGX_Provisioning_Certification_RootCA.pem

### D. User Authentication

During an authentication process, the enclave that is being used by the Authenticator Emulator receives a UID (provided by the Authenticator), an RP (provided by the client), and a Credential_ID and a challenge (provided by the RP). Using the `key-pair` file, it first checks if the credential exists and whether it was imported or needed to be derived (looking into the existence of a fifth element in the tuple). In the first case, the key pair is fetched from the file, and upon validation on the received UID and RP, it produces the desired assertion by making a signature on the challenge.

Otherwise, the credential must be derived. The received UID and RP is validated, through the `key-pair` file and using the Credential_ID the enclave can derive the key pair and answer the challenge.

## IV. Implementation

We developed our Authenticator Emulator for Linux systems. It uses the Linux `/dev/uhid` character device interface, which allows clients to identify and enumerate it. UHID (User-space HID I/O driver) is a kernel subsystem that allows userspace applications to implement the HID transport drivers without requiring extra kernel modules. This is suitable as it offers a direct implementation of the CTAP HID transport protocol entirely in userspace.

The emulated device is registered with an HID report descriptor that follows the FIDO Alliance HID usage page (`0xF1D0`), with usage `0x01` identifying it as a FIDO authenticator device. In the creation description are also included the two 64-byte reports: an input report for data sent from the authenticator to the host, and an output report for data received from the host. This 64-byte packet size is mandated by the CTAP HID specification and defines the fundamental transport unit for all CTAPHID messages.

With a software-emulated device ready to communicate, the next step is handling each received command. To exemplify this, consider the most common scenario: a browser initiating a FIDO2 credential registration or authentication flow.

The first command the Emulator receives is `CTAPHID_INIT`. The Emulator takes this opportunity to establish trust with an SGX host (defined by the user) by contacting the `/attest` endpoint and obtaining a `QUOTE`, which is verified to be properly signed and containing the expected `MRENCLAVE` value. If this attestation check fails, the initialization response returns an error and the entire flow is aborted before any sensitive operation takes place. This step binds the enclave's public key, which will be used to secure all subsequent communication.

Once the channel is established, the client issues `GetInfo` to find out what the authenticator can do. The emulator simply forwards this to the SGX host via `/getauthinfo`, using the enclave's public key to protect the request, and hands the response back to the client.

Now, having this completed, the next command is either the `MakeCredential` or the `GetAssertion`. Like in `GetInfo`, each of these operations has its own HTTP endpoint; the difference is that now the user interaction with the Emulator is required, as is necessary to insert the user attributes to compute their UID. In the enclave, these requests are received and the processes described above are carried out to produce a well-formed CTAP2 response. The browser can then forward this response to the relying party following the WebAuthn protocol.

The migration flow does not involve any CTAP interaction. Instead, communication occurs directly between the Authenticator Emulator, which collects the user's attributes, and the two credential management service instances involved in the transfer, as mentioned earlier.

We are currently developing the enclave code in order to implement several types of credentials. A topic that we are also designing is an efficient way to deal with a very large `key-pair` file, since the enclave memory is limited and beyond a certain threshold its performance reduces substantially due to encrypted page swapping.

To overcome this memory limitation, we propose a different storage model that contains N files instead of only one. In the registration, the enclave chooses in which of the N files to store the tuple by computing the modulus of an hash containing both the UID and the IK by N. When a migration occurs, the destination enclave also computes this identifying the respective UID file.

Making the hash include only the UID is meant to facilitate the credential exportation, as it is only necessary to access one file, while preserving anonymity by including the IK and reducing the credential search complexity by O(n/N).

## V. Security Vulnerabilities

Our approach has three identified vulnerabilities. The first is the trust on the Authenticator Emulator to behave properly. The answer here is that users may be able to select different Emulators, possibly from open-source code or trusted manufacturers.

The second is the resilience to destructive attacks against the files used by the SGX enclave. The corruption or destruction of these files voids all the credentials they have registered. Backups on SGX hosts are the obvious solution. But our credential migration also makes this attack less critical.

The third is DoS attacks with malicious registration requests for different RPs, which can cause the list of UID-RP mappings to grow endlessly, consuming enclave resources and potentially degrading performance. Limiting by UID is useless, since UID generation is not controlled. The only obvious solution for this is to implement some sort of accounting in the SGX host service.

## VI. Comparative Analysis With Related Work

This section presents a comparative analysis against existing credential management approaches: Hardware Keys, Platform Authenticators, Traditional Password Managers, Passkey Managers with Cross-Platform Synchronization (CXP), and FeIDo [15], a research prototype based on SGX and eID.

The comparison focuses on properties relevant to first-factor authentication, namely security, usability, compatibility, trust, and performance, as presented in Table I.

### A. Security and Trust

Related to security, hardware-backed solutions (hardware keys, platform authenticators, FeIDo, and our proposed approach) provide a stronger credential isolation and protection from OS compromise, while software-based solutions (traditional password and passkey managers) fail ensuring these types of guaranty. Within the SGX-based approaches, the proposed solution supports two credential models: derived credentials, where the private key is never stored but deterministically derived on demand from a hardware-bound identity key, offering the strongest isolation guaranty; and migrated credentials, which are imported from another SGX instance and stored within the enclave, maintaining hardware-backed protection while enabling cross-deployment portability.

Moving to the trust model, hardware keys trust only the token manufacturer (simplest trust model). Platform authenticators (Apple, Google, Microsoft) convert trust to platform's vendors, giving them much greater vendor lock-in and privacy issues, since they are now privy to all authentication activities. When it comes to software-based password management, users need the trust of the software, as security and privacy implications can be inconsistent between implementations and policies. FeIDo presents the most complex trust model because trust must be placed on Intel's SGX implementation, the software that hosts the enclave infrastructure, the enclave operations, and the eID issuer. Both FeIDo and the proposed approach benefit from a notable SGX property: the integrity of the enclave code can be cryptographically verified through remote attestation, meaning that any party can confirm that the software running inside the enclave has not been tampered with. This provides a verifiable basis for trust that is absent in software-only solutions. The proposed method mitigates FeIDo complexity by removing the dependency on the eID issuer but requiring trust in Intel SGX implementation (the service provider) and the Authenticator Emulator, which is responsible for collecting user attributes and mediating credential registration and authentication. Trust in the client browser, although not reflected in Table I, remains relevant for all solutions.

### B. Usability and Portability

Hardware keys and platform authenticators, though highly secure, do not allow the portability of credentials. Traditional password and passkey managers with strong encryption compatibility and CXP support ensure high portability through cloud synchronization or encrypted export, but sacrifice hardware-backed security. FeIDo provides a middle ground via eID-based credential derivation to offer medium portability without cloud synchronization, but with an added concern of having to interact with an eID for each authentication and the high initial setup complexity. The proposed approach achieves high portability through user-initiated, attestation-verified migration protocols between SGX instances, allowing credentials to be securely exported and imported across different deployments on demand, without sacrificing hardware-backed security guaranties. From a usability perspective, the solution's hard-coded Root CA is a primary concern; its expiration necessitates a code change that alters the `MRENCLAVE` measurement, ultimately requiring users to re-register their credentials with all Relying Parties.

### C. Compatibility

The compatibility analysis exposes a critical limitation in current TEE-based approaches. While hardware keys, platform authenticators, and passkey managers support a fully compliant CTAP protocol that works in tune with native applications, FeIDo rejects that standard and instead uses a browser

TABLE I
COMPARISON OF CREDENTIAL MANAGEMENT APPROACHES

| **Property** | **Hardware Keys** | **Platform Auth.** | **Trad. Passwords Mgrs** | **Passkey Mgrs (CXP)** | **FeIDo** [15] | **Proposed** |
|---|---|---|---|---|---|---|
| **Security Properties** | | | | | | |
| Credential Isolation | High (Sec. Elem.) | Med-High (TEE/TPM) | Low (SW) | Low (SW) | High (SGX) | High (SGX) |
| Phishing Resistance | ✓ | ✓ | × | ✓ | ✓ | ✓ |
| Protection from OS | ✓ | ✓ | × | × | ✓ | ✓ |
| **Usability** | | | | | | |
| Portability | None | None | High | High | Medium | High |
| Multi-device Access | Physical transfer | × | ✓(sync) | ✓(export) | Limited (eID req.) | ✓ |
| Recovery Mechanism | None | Platform dep. | Cloud sync | Encrypted export | eID derivation | User-centric migration |
| Setup Complexity | Low | Low | Low | Medium | High | Medium |
| **Compatibility** | | | | | | |
| CTAP Protocol | ✓ | ✓ | N/A | ✓ | × | ✓ |
| Native Apps | ✓ | ✓ | ✓ | ✓ | × | ✓ |
| OS Independence | ✓ | × | ✓ | ✓ | Partial | Partial |
| Client Hardware Req. | ✓(token) | ✓(TPM/TEE) | × | × | Optional | Optional |
| **Trust Model** | | | | | | |
| Trust Requirements | Token manufact. | Platform vendor | Service prov. | Serv.+recipient | SGX+Serv.+eID issuer | SGX+Service prov. |
| Privacy | Medium (attest.) | Medium (OS telem.) | Variable | Variable | High (anon.+SGX) | High |
| **Performance** | | | | | | |
| Auth./Reg. Latency | Low | Low | Low | Low | Medium (enclave) | Medium (enclave) |

extension. This architectural choice restricts FeIDo to web-based authentication scenarios, leaving out native applications, command-line tools, and OS-level authentication. On the other hand, the proposed approach relies on the UHID interface, which is only supported in Linux systems, contrasting with the full OS independence of hardware keys and the other software-based solutions. However, this does not mean that we cannot implement our Mediator in other operating systems, possibly using other low-level mechanisms.

### D. Privacy and Performance

Related to privacy, for hardware keys this is not a concern (a potential cross-site tracking through attestation certificates that declare the vendor is tackled by replicating the certification credentials in batches of devices) and platform authenticators may expose authentication patterns through OS-level telemetry and credential synchronization services (iCloud Keychain, Google Password Manager). Traditional password and passkey managers show variable privacy depending on service provider policies and cloud synchronization practices. FeIDo achieves the strongest through anonymous credential mechanisms that enable attribute verification without revealing identity. The proposed approach achieves high privacy, as the credential keys are protected within the SGX enclave and never exposed to the service layer.

Regarding performance, hardware keys and software-based solutions have low latency, while both FeIDo and our approach incur a medium latency overhead due to the cost of network communication and enclave context switches and operation.

### E. Key findings and Motivation

This comparative review reveals a missing link in alternative credential management methodologies: no approach could achieve an adequate combination between strong TEE-backed security and high portability while remaining fully compatible with CTAP protocols. Hardware keys provide great security and CTAP support without any credential portability. Although platform authenticators have TPM protection, they create full vendor lock-in with no portability. Passkey managers need to leverage CXP for portability, but operate entirely in software without hardware security guaranties. FeIDo is the closest to our research objectives, using SGX for credential protection and recovering via eID-based derivation although also fundamentally abandoning CTAP support in favor of browser add-ons, as with the complexity of eID usage and integration.

*1) The Attestation Gap:* Hardware authenticators support attestation mechanisms that allow relying parties to verify security characteristics, while software-based FIDO2 implementations cannot provide equivalent assurances. FeIDo demonstrates that SGX-based virtual authenticators can offer attestation capabilities, but the abandonment of CTAP protocol creates a deployment barrier that limits practical adoption.

*2) The Migration Gap:* Overall, FeIDo remains the closest work to our solution, yet beyond the concerns already discussed, it presents a more fundamental limitation: credential migration is not addressed. This raises significant availability concerns, particularly at scale. Consider the worst case: a large number of users derive credentials tied to a specific SGX processor, and that processor fails or becomes unavailable. Since credentials are always derived from hardware-bound secret material, they cannot be recovered on different hardware, leaving all affected users locked out. This represents a critical bottleneck in any derivation-only approach.

Keeping this in mind, the proposed solution supports user-initiated migration, which means that credentials are not permanently tied to a single hardware instance. If a deployment becomes unavailable, either due to hardware failure or a denial-of-service attack, users can switch to a different SGX instance to which they have migrated their credentials before losing access. This decoupling between credentials and hardware is a deliberate design choice that FeIDo, by relying solely on derivation, cannot offer, but is crucial in order to scale the solution.

*3) The Trust Asymmetry:* Finally, the trust asymmetry in remote enclave architectures presents an unsolved challenge. As discussed earlier, credential management with remote SGX enclaves faces a limitation: while the enclave can attest to its security properties, typical client devices without SGX support cannot provide equivalent cryptographic assurances. This asymmetry complicates secure credential provisioning and migration protocols, as the enclave must rely on traditional authentication mechanisms (user attributes, passwords, cryptographic keys) to verify user authorization.

## VII. Conclusions

We presented an SGX-based FIDO2 credential manager driven by a CTAP-enabled software Authenticator (Emulator). This system reduces barriers to strong authentication by removing the need for physical tokens while maintaining similar security guaranties. The system enables the Emulator to verify the correctness of credential management code (SGX enclave) through remote attestation, eliminating blind trust in service providers. It also supports multiple server instances to be used by each Emulator and the migration of credentials among them to improve availability, but without relaxing on security; credentials can only be migrated between equal enclave instances, properly attested. The leverage of SGX's security properties, particularly memory encryption and isolated execution, can effectively mitigate relevant attack vectors against user credentials. Thus, in essence, we built a centralized credential manager with replication capabilities, whose infrastructure provider does not need to be trusted, while maintaining the usability advantages of traditional password managers and the security benefits of FIDO2.

As future work, we intend to tackle the identified DoS vulnerability, improve the Authenticator Emulator to use a local vault (e.g., TPM-based) to store a UID protected by a secret or biometrics, and study how we could use an enclave `QUOTE` to leverage the SGX certification that exists for hardware tokens.

## Acknowledgment

This work was supported by the Foundation for Science and Technology (FCT) through contract doi.org/10.54499/UID/00127/2025.

## References

[1] D. Temoshok, J. L. Fenton, and Y.-Y. C. et al., "Digital Identity Guidelines: Authentication and Lifecycle Management," National Institute of Standards and Technology, Tech. Rep. NIST SP 800-63B-4, Jun. 2025, DOI: 10.6028/NIST.SP.800-63B-4.

[2] S. Das, A. Dingman, and L. J. Camp, "Why Johnny Doesn't Use Two Factor: A Two-Phase Usability Study of the FIDO U2F Security Key," in *22nd Int. Conf. on Financial Cryptography and Data Security (FC 2018), LNCS 10957*, Nieuwpoort, Curaçao, Feb. 2018, pp. 526–543, DOI: 10.1007/978-3-662-58387-6_9.

[3] K. Reese, T. Smith, J. Dutson, J. Armknecht, J. Cameron, and K. Seamons, "A Usability Study of Five Two-Factor Authentication Methods," in *Fifteenth Symp. on Usable Privacy and Security (SOUPS 2019)*, Santa Clara, CA, USA, Aug. 2019, pp. 357–370, DOI: 10.1145/3460120.3484746.

[4] J. Colnago, S. Devlin, M. Oates, C. Swoopes, L. Bauer, L. Cranor, and N. e. a. Christin, "It's not actually that horrible: Exploring Adoption of Two-Factor Authentication at a University," in *CHI Conference on Human Factors in Computing Systems (CHI'18)*, Montreal, Canada, Apr. 2018, DOI: 10.1145/3173574.3174030.

[5] V. Costan and S. Devadas, "Intel SGX Explained," Cryptology ePrint Archive, Paper 2016/086, 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[6] World Wide Web Consortium (W3C), "Web Authentication: An API for accessing Public Key Credentials Level 2," W3C Recommendation, Apr. 2021. [Online]. Available: https://www.w3.org/TR/2021/REC-webauthn-2-20210408/

[7] FIDO Alliance, "Client to Authenticator Protocol (CTAP)," Review Draft, Oct. 2025. [Online]. Available: https://fidoalliance.org/specs/fido-v2.3-rd-20251023/fido-client-to-authenticator-protocol-v2.3-rd-20251023.html

[8] M. Barbosa, A. Boldyreva, S. Chen, and B. Warinschi, "Provable Security Analysis of FIDO2," in *41st Annual Int. Cryptology Conf. (CRYPTO 2021), LNCS 12827*, Aug. 2021, DOI: https://doi.org/10.1007/978-3-030-84252-9_5.

[9] J. Kunke, S. Wiefling, M. Ullmann, and L. L. Iacono, "Evaluation of Account Recovery Strategies with FIDO2-based Passwordless Authentication," in *Open Identity Summit 2021, LNI*, May 2021. [Online]. Available: https://dl.gi.de/items/0d2e6c0b-0bd7-4d32-9e0b-7a0926a2b713

[10] S. G. Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel, "Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication," in *IEEE Symp. on Security and Privacy (SP)*, San Francisco, CA, USA, May 2020, DOI: 10.1109/SP40000.2020.00047.

[11] M. Barbosa, A. Cirne, and L. Esquível, "Rogue key and impersonation attacks on FIDO2: From theory to practice," in *18th Int. Conf. on Availability, Reliability and Security (ARES'23)*, Benevento, Italy, Aug. 2023, DOI: 10.1145/3600160.3600174.

[12] R. Hussein, "Exploration of the Security and Usability of the FIDO2 Authentication Protocol," Ph.D. dissertation, Univ. of Maryland, 2022, DOI: 10.13016/wkfq-edas.

[13] M. Casagrande and D. Antonioli, "CTRAPS: CTAP Client Impersonation and API Confusion on FIDO2," in *IEEE 10th European Symposium on Security and Privacy (EuroS&P)*, Venice, Italy, Jun. 2025, OID: 10.1109/EuroSP63326.2025.00063.

[14] C. Shepherd, R. N. Akram, and K. Markantonakis, "Remote Credential Management with Mutual Attestation for Trusted Execution Environments," in *12th IFIP WG 11.2 Int. Conf. on Information Security Theory and Practice (WISTP 2018)*, Brussels, Belgium, Dec. 2018, DOI: 10.1007/978-3-030-20074-9_12.

[15] G. Tsudik, T. Frassetto, D. Gens, C. Liebchen, and A.-R. Sadeghi, "FeIDo: Recoverable FIDO2 Tokens Using Electronic IDs," in *ACM SIGSAC Conf. on Computer and Communications Security (CCS'22)*, Los Angeles, CA, USA, Nov. 2022, DOI: 10.1145/3548606.3560584.