Protecting FIDO Extensions Against Man-in-the-Middle Attacks

Andre Büttner(✉) and Nils Gruschka

University of Oslo, Gaustadalléen 23B, 0373 Oslo, Norway
{andrbut,nilsgrus}@ifi.uio.no

Abstract. FIDO authentication has many advantages over password-based authentication, since it relies on proof of possession of a security key. It eliminates the need to remember long passwords and, in particular, is resistant to phishing attacks. Beyond that, the FIDO protocols consider protocol extensions for more advanced use cases such as online transactions. FIDO extensions, however, are not well protected from Man-in-the-Middle (MitM) attacks. This is because the specifications require a secure transport between client and server, but there exists no end-to-end protection between server and authenticator.

In this paper, we discuss MitM scenarios in which FIDO extensions may be intercepted. We further propose an application-layer security protocol based on the CBOR Object Signing and Encryption (COSE) standard to mitigate these threats. This protocol was verified in a formal security evaluation using ProVerif and, finally, implemented in a proof-of-concept.

Keywords: Security · FIDO · WebAuthn · CTAP2 · COSE · Encryption

1 Introduction

In today's digital era, almost everybody is used to log in to a website with a password. Although passwords are easy to use, they have many disadvantages in terms of security. They are often easy to guess or sometimes publicly disclosed after a data breach. Furthermore, passwords are vulnerable to phishing attacks. Therefore, many services already implement multi-factor authentication.

FIDO authentication is a relatively young technology that intends to overcome the disadvantages of passwords. The basic idea behind it is to use an authenticator device as a more secure authentication factor, either in addition to or even as a replacement for passwords. A feature that is rarely used yet but may become important soon are FIDO extensions. These allow for more advanced functionality beyond simple authentication. FIDO authentication may, for example, be used to confirm online purchases or banking transactions. Initially, a FIDO Transaction Confirmation extension was proposed, which includes a human-readable text representation of a transaction as an extension [13]. This extension, however, was never implemented and already became deprecated. It

A. Saracino and P. Mori (Eds.): ETAA 2022, LNCS 13782, pp. 70–87, 2023.
https://doi.org/10.1007/978-3-031-25467-3_5

is replaced by the more recent Secure Payment Confirmation [27]. We expect to see more different kinds of extensions like this in the future.

However, the FIDO specifications do not provide any specific protection for FIDO extensions. Since extensions can contain very sensitive information, it should be ensured that attackers cannot intercept or manipulate this information. There are several possibilities for attackers to act as Man-in-the-Middle (MitM). The FIDO protocols only protect the integrity of messages from the authenticator to the server. The integrity of messages from the server cannot be checked by the authenticator. While this is not necessary for basic authentication, this may be crucial for certain extensions. Also, confidentiality cannot be guaranteed as there is no encryption on the application layer.

To mitigate the risk of manipulation or disclosure of FIDO extensions, we propose to apply authenticated encryption to FIDO extensions. In this paper, we provide the following contributions:

1. An overview of different MitM attack scenarios against FIDO extensions.
2. A proposal for a security protocol to protect FIDO extensions.
3. A formal security verification of this protocol.
4. A proof-of-concept implementation.

The remainder of this paper is structured as follows. Section 2 gives an overview of FIDO authentication and the COSE protocol. In Sect. 3 related literature on FIDO is presented. The attack model addressed in this paper is explained in Sect. 4. In Sect. 5 we specify our proposed security protocol, which is evaluated in Sect. 6. In Sect. 7 we describe a proof-of-concept implementation. A discussion of the proposed solution is provided in Sect. 8. Our findings are summarized in Sect. 9 along with a brief outlook on future work.

2 Background

In this section we firstly provide some background information on FIDO authentication. Afterwards the CBOR based COSE protocol is described.

2.1 FIDO Authentication

The Fast IDentity Online (FIDO) Alliance is publicly active since 2013 [16]. Today it includes members from several popular Internet companies. Their main objective is to provide industry standards for using authenticators to authenticate against web applications either as a single factor (password-less) or as an additional factor (2FA/MFA). There are two different types of authenticators. *Roaming authenticators* are external security tokens (for example a YubiKey) that can be connected via USB, Bluetooth-Low-Energy (BLE) or Near-Field-Communication (NFC). In contrast to this, *platform authenticators* are integrated into client devices like computers and smartphones.

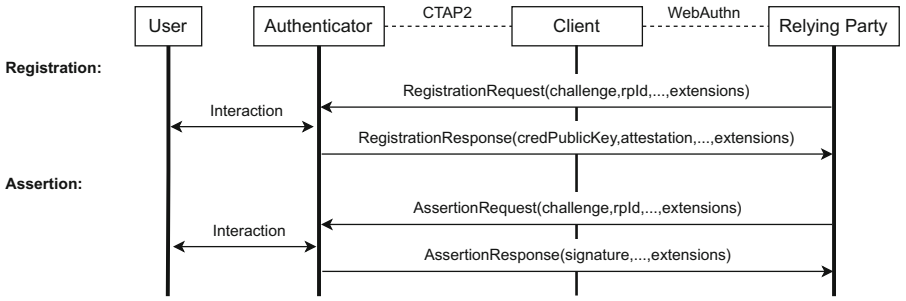


Fig. 1. FIDO authentication overview.

In this paper, we mainly focus on FIDO2, which consists of the Web Authentication (WebAuthn) API and the Client-to-Authenticator-Protocol 2 (CTAP2) [15]. WebAuthn has become a W3C standard [22] and defines a JavaScript API and data structures that can be used to create credentials and get assertions from the authenticator. CTAP2 defines the protocol between the client platform and the authenticator.

For FIDO authentication, security and trust are based on public key cryptography. Figure 1 gives a brief overview of the different roles and messages that are specified for FIDO2 authentication. At first, an authenticator (e.g. a security token) needs to be registered at a web service, the so-called relying party (RP). When a user registers at a RP, the RP firstly sends a registration request to the authenticator which includes a random nonce (challenge), its identifier (rpId), and further parameters. The authenticator creates a credential key pair and shares its public key with the RP by sending a registration response. In addition, the authenticator may include an attestation certificate that verifies the origin of the authenticator by a certified manufacturer. For this purpose, the FIDO Alliance provides a public service called *Metadata Service* [14], which contains a list of vendors, their public keys and their certification levels. During authentication, the RP creates another challenge value and sends it to the authenticator in an assertion request. This challenge is signed by the authenticator, along with other parameters, using the private key of the credential that was previously registered with the RP. Using the corresponding public key, the RP can verify the signature of the assertion response. Both for registration and authentication, the user needs to interact with the authenticator, e.g., by pressing a button. For more security, the user can enter a PIN or interact with a biometric scanner, which provides an additional authentication factor.

The FIDO specifications leave room for additional features by using protocol extensions. Extensions sent by the RP to the authenticator are called *input extensions* and extensions from the authenticator to the RP *output extensions*. Furthermore, it is distinguished between *client extensions* and *authenticator extensions*. In this paper, we focus on authenticator extensions, i.e., those that are processed by the authenticator. Several different types of extensions have been

proposed (see e.g. [20]), however, almost none of these has been implemented yet. The Secure Payment Confirmation [27] is a new W3C specification and describes a payment extension for FIDO authentication. It is a good example of more advanced applications of FIDO authentication. However, it must be kept in mind that such an extension also requires high security standards.

2.2 COSE

The *CBOR Object Signing and Encryption* (COSE) [33] protocol aims to provide a standard for exchanging signed and encrypted data in the *Concise Binary Object Representation* (CBOR) [6] format. CBOR is a binary data format that is particularly useful for low-resource devices due to its lightweight and efficient design. Among other things, it is used in the CTAP2 protocol. Data can be structured into maps and arrays of various types. Consequently, JSON objects can be easily converted to CBOR objects, which makes it also usable from a developer's perspective. Furthermore CBOR provides features like tags and flexible map and array lengths.

COSE is basically adapted from the JavaScript Object Signing and Encryption (JOSE) protocol. It defines data structures for exchanging data that is signed, encrypted or authenticated (MAC). COSE objects carry the payload together with additional information about the keys and algorithms that are used. A COSE message is composed of a CBOR array that contains a protected header, an unprotected header, the payload and depending on the type additional values like the signature or the message authentication tag. A protected header is a CBOR encoded map of certain header values. It is used as input in addition to the actual payload for cryptographic functions, e.g., as additional authenticated data (AAD) when using authenticated encryption or as input for the signature. The unprotected header is a map that contains further header values, which in contrast to the protected header are not cryptographically bound to the payload or signature. COSE signature messages may contain one (`COSE_Sign1`) or multiple signatures (`COSE_Sign`). Encryption messages can be intended for one recipient (`COSE_Encrypt0`) or for multiple recipients (`COSE_Encrypt`). Respectively, there are also two different COSE MAC structures.

The COSE protocol does not specify, how keys are negotiated by the different parties. However, it defines a *COSE Key* structure which contains all necessary information for a key. This can be useful, e.g., for storing the key or for sending it to another party in a standardized manner. For example, the FIDO2 protocols make use of the COSE Key format to send the public key from the authenticator to the RP. There currently exists a draft for a COSE based *Ephemeral Diffie-Hellman Over COSE* (EDHOC) protocol to provide additional features like key negotiation [34], which, however, is not standardized yet.

3 Related Work

Since FIDO authentication is a fairly new topic, research on the subject is still very limited. We therefore provide a brief overview of the related literature.

There has been quite some research on the usability of FIDO authentication [28,29]. One of the major concerns by the users is the account recovery. If the authenticator gets lost, there must be some way to regain access to the account. At the same time, the recovery option should not reduce the security. As a solution, an enhancement for the protocol was proposed that enables the use of a backup key that only needs to be configured once in the beginning [17]. Furthermore, a study on different account recovery approaches was conducted to compare them in terms of usability, deployability and security [23]. Other researchers applied formal methods to analyse the security of the FIDO protocols [3,11]. In particular, there still seems to be a lack of research that focuses on the CTAP2 protocol.

There has also been done little research specifically on the security of FIDO extensions. For example, it was proposed to use structured data formats for FIDO Transaction Confirmation to facilitate the validation of transaction information by the authenticator thereby making it more secure [8]. Furthermore, some researchers have pointed out that the FIDO Transaction Confirmation extension is vulnerable to manipulation. They propose to let an RP sign the transaction information, which can be validated on the client-side in a trusted environment [37,38]. However, they do not point out how the authenticity of the public key is guaranteed. In addition to this, we see further risks. If FIDO extensions can be manipulated, they can also be eavesdropped in similar attack scenarios. Therefore, confidentiality should be equally considered alongside integrity and authenticity.

4 Attacker Model

The WebAuthn standard [22] requires a so-called secure context, which includes the use of HTTP over TLS (HTTPS). This ensures that the client can verify the authenticity of a web server. Yet, there are more components involved that can interfere with FIDO messages beyond client and server. We therefore argue that HTTPS does not provide sufficient protection for FIDO messages at all. FIDO authentication can involve several different intermediaries between the RP and the authenticator. These include (1) web proxies between the client and the RP, (2) the client application, (3) intermediary processes on the client platform and (4) hardware between a roaming authenticator and the client device. Thus, there is a significantly large attack surface, as illustrated in Fig. 2.

Plain authentication—also referred to as *entity authentication*—without any extensions is not likely at risk, because the protocol is designed not to contain sensitive information. Also, it is resistant against manipulation through the challenge-response protocol. However, authentication that involves extensions exchanged between the RP and the authenticator may contain valuable information, e.g., personal data, transaction details or other sensitive information. Such information could be obtained or manipulated by an adversary. In the following, we elaborate on the four different MitM scenarios in more detail.

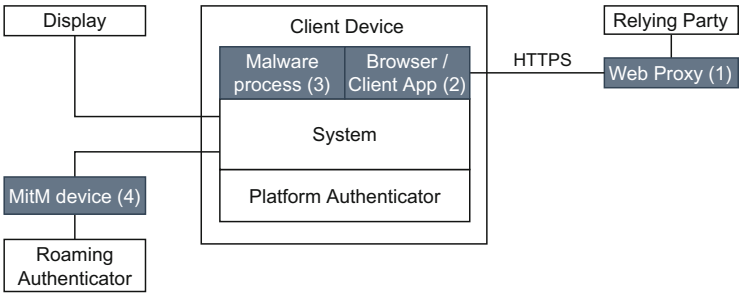


Fig. 2. Attack surface: possible points of interception are highlighted.

4.1 Vulnerable Web Intermediaries

Distributed systems like web applications typically include intermediaries such as content delivery networks (CDN), load balancers or web application firewalls (WAF). The secure context requirement mentioned above can only be verified for the connection between a browser and the next HTTP entity. As a consequence, it cannot be guaranteed that intermediaries communicate with other intermediaries or the server via HTTPS. Apart from that, HTTPS is only protecting on the transport layer. Web intermediaries usually operate on the application layer and therefore need to access HTTP messages including the body. Consequently, they are able to read FIDO messages in clear text.

If a proxy behaves maliciously, this can have severe security implications. A proxy could be misused to intentionally read FIDO messages and disclose sensitive information. Beyond that, a malicious proxy could manipulate extensions. There is no integrity check considered for FIDO messages from the RP to the authenticator. Depending on the type or use of an extension, only a user may notice the manipulation through manual inspection. For messages from the authenticator to the RP, any manipulation will be detected by the RP, since the authenticator data including extensions are signed. In any case, there is still the risk of information disclosure.

Another concern with HTTP intermediaries is the possibility of attacks that result from the semantic gap of the HTTP protocol [9]. In particular, cache poisoning vulnerabilities could be exploited to disclose FIDO messages. This can be realized by various techniques like request smuggling [26] or web cache deception [18].

4.2 Compromised Client Application

The client application on the user's device may be running in a browser as a JavaScript application or a native mobile application. Both browser clients and native mobile applications often use third-party libraries. If not checked properly, such libraries can include malicious code [2,39]. Another possibility to compromise the client application is to exploit cross site scripting (XSS)

vulnerabilities in a JavaScript application to inject code that intercepts FIDO messages and modifies extensions or forwards them to an untrustworthy third party.

4.3 Malware on the Client Device

Malware can pose a further threat to FIDO extensions. An attacker may be able to install malicious software on a user's client device through an email attachment or some other exploit. By intercepting inter-process communication (IPC) or accessing memory of other processes, the malware could read or manipulate FIDO messages. Moreover, it could bypass security measures by the browser and system and send its own FIDO assertions to the authenticator. It was already shown that this can cause a user to confirm a malicious transaction [7]. In addition, there may be specific types of viruses targeting browsers on client devices. By this, a browser may be corrupted in a way that it can be controlled by an attacker, which is also known as Man-in-the-Browser (MitB) attacks [10]. Beyond that, platform authenticators are generally at risk of behaving unintentionally when they are affected by malware. This can be mitigated with the use of trusted platform modules (TPM), which make sure that secret keys are not disclosed. Nevertheless, exploits against extensions remain a threat.

4.4 MitM Between Client Device and Authenticator

With respect to roaming authenticators, an attacker may try to intercept the connection between the client device and the authenticator. This is certainly a more difficult attack, since an attacker needs physical access to the user's devices. One could argue that the security of the FIDO device is completely compromised in that case and other MitM countermeasures would be pointless. However, this is only true for authenticators that just require a button press and not when the authenticator uses a more secure verification method such as biometrics.

Even if an authenticator uses a verification method like biometrics, an attacker may still be able to intercept the connection and eavesdrop or manipulate extensions. For example, there are known MitM attacks against Bluetooth [24,35]. NFC is very unlikely to be intercepted without the owner's awareness. But still, a potential attack against NFC has been demonstrated [1].

5 Protocol Design

As shown in the previous section, FIDO messages can indeed be vulnerable to several attacks. Extensions may include sensitive information and are at risk of being modified by or disclosed to unauthorized parties. Considering the large attack surface, we see the necessity to apply further measures. In this section, we present our proposed protocol to protect FIDO extensions.

5.1 Authenticated Encryption

Sensitive FIDO extensions should provide secure properties such as confidentiality, integrity and authenticity. Messages sent from the authenticator to the RP are already signed and, thus, do not require any additional authentication. However, messages from the RP to the authenticator are neither signed, nor authenticated by any means. Xu *et al.* [37] suggest that the relying party shall sign extensions. In their approach the verification of the signature is done on the client device and the public key for verifying the signature is given by the TLS connection. We, however, want to enable the authenticator itself to validate the authenticity of extensions from the RP.

Since signatures do only provide integrity and authenticity, but no confidentiality, we propose to fulfill all these properties by using authenticated encryption (AE) instead. For this the RP and the authenticator must firstly agree upon a shared secret during registration. After that they can derive key material from the shared secret and use AE algorithms such as AES-GCM to encrypt and authenticate extensions that are included in assertion messages.

There can be multiple authenticators registered with one user account on the RP. However, there will be a different shared key between the RP and every authenticator. The RP does not know which registered authenticator will be used for the assertion. Therefore we need to apply *key wrapping*. This means that the actual extension is encrypted with a newly created content encryption key. This key is then encrypted with the shared key and appended to the message for each authenticator. For encrypting the extensions in the assertion response by the authenticator the shared key can be used directly, because the message is only intended for the RP. This is formalized in our model in Sect. 6.2.

5.2 Key Exchange

Encrypting FIDO extensions requires the RP and the authenticator to exchange keys in advance. Normally, a public-key infrastructure (PKI) is used to create certificates which provide trust and authenticity for exchanged keys. Hardware tokens, however, are very limited and likely not powerful enough in terms of storage and computation to handle certificate chains. Since they operate offline, there is also no possibility for them to directly interact with certificate authorities over the network (e.g. to check on certificate revocations). This is different for other types of authenticators with more computing resources and networking capabilities. However, we want to address all types of authenticators with our solution. Because of this, we consider the *trust-on-first-use* authentication scheme [36]. This means that we trust the first connection between authenticator and RP not being intercepted by an adversary.

To generate a shared secret, the RP and the authenticator perform a Diffie-Hellman key exchange (DHKE) during the registration. The RP includes the first part of the DHKE as registration input extension. The authenticator generates and stores the shared secret from the DHKE and sends a registration response to the RP, which includes the second part of the DHKE as registration output

extension. Finally, the RP generates the shared secret from the DHKE and stores it together with the newly registered credential.

The (unauthenticated) DHKE is known to be secure against eavesdropping, but vulnerable to active MitM attacks. Authenticators use attestations that should be validated by RPs to create a certain amount of trust. If properly done, this can mitigate active MitM attacks. However, for higher security, it is important that the authenticator includes both parts of the DHKE in its attestation signature, as shown in Sect. 6.1.

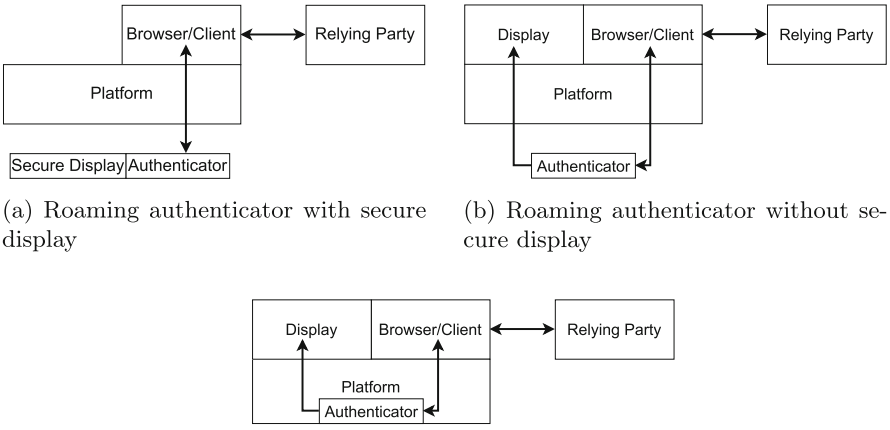
5.3 Data Format

A further important aspect is the format used to exchange the encrypted data along with required metadata like an input vector (IV) and the algorithm used. FIDO authenticator extensions must be provided in the CBOR format. As described in Sect. 2.2, the accompanying protocol for signature and encryption is COSE. Since the public key from the authenticator is transmitted as a COSE key, authenticators and RPs are both supporting parts of the COSE protocol already. The COSE standard supports encryption for single and multiple recipients, and thus provides all functionality needed for the proposed protocol. Our suggestion is therefore to embed extensions in COSE structures. The COSE key format can also be used to encode the DH public keys that are exchanged during the registration to generate the shared secret.

5.4 Displaying User Information

When encrypting extensions, we still need to be able to display information, such as transaction information, to the user in a secure manner. This is the key aspect of the What-You-See-Is-What-You-Sign principle [25]. The different possible architectures with FIDO authenticators are displayed in Fig. 3. Ideally, an authenticator should provide a secure display (Fig. 3a). However, there are no roaming authenticators with a display on the market yet. In most cases, the client platform would be responsible for displaying the information to the user. With our approach, the client will not be able to decrypt the extensions on transit. Therefore, the authenticator firstly needs to decrypt the extensions and then forward the user information to the client display on a secure path (Fig. 3b). For this the platform should provide appropriate measures to ensure that there is no interception possible when displaying the information to the user.

Platform authenticators are integrated into computers or smartphones. Since these already have a display, platform authenticators can provide the decrypted user information instantly to the platform without an intermediary connection (Fig. 3c). FIDO authentication is already supported by most platforms like Windows [21], MacOS/iOS [30] and Android [19]. The platform itself must ensure that the information that is displayed to the user has not been modified by another malicious process. When the client device is responsible for displaying the information, there are further risks like UI deception attacks [4,12], which,



(a) Roaming authenticator with secure display

(b) Roaming authenticator without secure display

(c) Platform authenticator

Fig. 3. Architectures for different types of authenticators.

however, must be taken care of by the operating system. The same applies when a roaming authenticator without a secure display is used.

6 Security Evaluation

ProVerif [5] is a common protocol verifier that uses Horn-clause based representations of a protocol and the applied π-calculus for process verification. A formal model of the protocol and the security properties to be tested are defined in input files. The output are the test results indicating whether the defined security properties are met. If a test fails, a possible attack trace is provided. This tool has been used to conduct a security evaluation of our protocol.

Formal models have been created for the key exchange during the registration of an authenticator and for the exchange of encrypted and authenticated extensions. A basic description of the models, some code excerpts and the results are given below. For more details, the sources and results of the evaluation can be found in our Github repository[1].

6.1 Key Exchange

In our protocol, a DHKE is performed to generate a shared secret on the relying party and the authenticator. Even though a client can verify the TLS certificate of a RP, we assume that an authenticator cannot validate the authenticity of a RP. However, we consider that a RP requires attestation from the authenticator

[1] https://github.com/Digital-Security-Lab/protecting-fido-extensions-proverif.

and that it verifies the attestation signature with a known and trusted public key. This serves to validate the authenticity of a DH key that is received by the RP to compute the shared secret.

One obvious security requirement is that the shared secret is kept secret and cannot be obtained by an attacker. This is defined by the following two queries:

```
query attacker(computeSecret(publicKey(dh_priv_AU),
    dh_priv_RP)).
query attacker(computeSecret(publicKey(dh_priv_RP),
    dh_priv_AU)).
```

Further, we check the authenticity by verifying that a key exchange is only performed if both authenticator and RP have generated the same secret:

```
query x:G; inj-event(sharedSecretRP(x))
    ==>inj-event(sharedSecretAU(x)).
```

For the key exchange, two different models of the protocol were created, because the first model did not pass the verification.

Protocol Model 1. In this model, the authenticator creates an attestation signature including the nonce, the credential public key and the output extension containing its DH key:

```
sign((nonce,credPubKey,dh_pub_AU),privKeyAttestation)
```

This signature is verified by the RP against the original nonce, the credential public key and the extension with the DH key by the authenticator:

```
checksign((nonce,credPubKey,dh_pub_AU),signature,
    pubKeyAttestation)
```

When verifying this model with ProVerif, the authenticity test fails. The detailed output of ProVerif contains a trace where an attacker intercepts a registration request by the RP and replaces the DH key of the RP with its own key. Because of this, the authenticator will compute a different shared secret than the RP. As a consequence, the authenticator cannot authenticate or decrypt assertion extensions from the RP, but from the attacker. However, the attacker cannot gain much from this, except for causing a denial of service. Nonetheless, this attack should be avoided.

Protocol Model 2. In the second model, the authenticator includes both DH keys in the attestation signature, so the RP can verify that the same shared secret is computed on both ends. From a theoretical perspective, it would be enough to only modify the signature. However, in practice the protocol proposed here should be compatible with the FIDO standards. Therefore the authenticator will have to include both its own DH key and the DH key from the RP in the output extensions, so both keys are implicitly included in the signature:

```
sign((nonce,credPubKey,dh_pub_AU,dh_pub_RP),
    privKeyAttestation)
```

The same is true when verifying the signature. In particular, the RP must check the signature with its own generated DH key and the authenticator's key:

```
checksign((nonce,credPubKey,dh_pub_AU,dh_pub_RP),signature
    ,pubKeyAttestation)
```

With this model all tests succeed and the secrecy of the shared secret is guaranteed as well as the authenticity of the public keys that were exchanged. We therefore consider this model in our final solution.

6.2 Encrypted Assertion Extensions

The second critical part of the proposed protocol is the exchange of encrypted and authenticated extensions between RP and authenticator during assertions. We make the following assumptions. First, the authenticator is successfully registered with the RP. This means that the RP has the credential public key of the authenticator to verify its signature and both authenticator and RP have exchanged a shared secret and derived from it a shared key. Second, while replay attacks against the RP are prevented by the nonce, replay attacks against the authenticator are not.

A security requirement here is the secrecy of input and output extensions, which is defined in the following two queries:

```
query attacker(AssertionInputExtensions).
query attacker(AssertionOutputExtensions).
```

In addition, the authenticator and the RP should both only accept authenticated extensions. An attacker must not be able to forge or manipulate extensions in a way that they are processed by either of them. As mentioned above, we assume an attacker to be able to replay assertion messages to the authenticator, but not to the RP. Therefore only events in connection with output extensions are defined as injective events:

```
query x:bitstring; event(receiveInputExtensionsAU(x))
    ==>event(sendInputExtensionsRP(x)).
query x:bitstring; inj-event(receiveOutputExtensionsRP(x))
    ==>inj-event(sendOutputExtensionsAU(x)).
```

This time only one model had to be created. In this model, the RP uses key wrapping to transmit a content encryption key together with the encrypted input extensions:

```
new cek:key;
let inputExtensions_enc = senc(AssertionInputExtensions,
    cek) in
let cek_enc = senc(key2Bitstring(cek), sharedKey) in
out(c,(nonceRP,cek_enc,inputExtensions_enc))
```

The authenticator, on the other hand, uses the shared key directly to encrypt the output extensions:

```
let outputExtensions_enc = senc(AssertionOutputExtensions,
    sharedKey) in
```

The evaluation of this model with ProVerif indicates that the expected security requirements are met and no attacks have been found. Hence, with this model, we can successfully exchange FIDO extensions while preserving their confidentiality, integrity, and authenticity.

7 Implementation

A proof-of-concept (PoC) application has been developed to demonstrate how to implement essential parts of the proposed protocol. The sources and further instructions can be found in a Github repository[2]. Since FIDO keys can be, e.g., USB devices with very limited resources, it was decided to provide a test application using the C programming language and libraries that are optimized for embedded devices. To implement the protocol, a CBOR library is needed, which is already included in each FIDO component, as it is required for implementing the basic FIDO protocols. Moreover, a COSE library is needed. Since we could not find a useful implementation, we developed an open-source COSE library[3] based on the RFC 8152 standard [33]. At the time of writing, this library is still a work in progress, but already provides everything needed for the proposed protocol. Finally, additional crypto libraries may be needed to do certain operations such as generating private and public keys, to compute the shared secret from the DHKE and to derive key material using e.g. a Hash Key Derivation Function (HKDF).

The PoC application is meant to demonstrate the parts that have to be implemented in addition to the FIDO protocols. Basic features such as credential creation, attestation and signature verification are therefore not included. For the DHKE, elliptic curve key pairs were used. The authenticated encryption is done using AES-GCM with a 128-bit key. In the example application, it is firstly shown how the RP and the authenticator exchange a shared secret. The RP creates the first part of the DHKE, which is encoded in a COSE Key structure and transmitted to the authenticator. The authenticator then creates the second part of the DHKE, computes the shared secret and derives from it a 128-bit key using a HKDF with SHA-256 as underlying hash function. Subsequently, the RP receives the second part of the DHKE (in a real world application together with the first part of the DHKE as discussed in Sect. 6.1) and analogously computes the shared secret and derives from it a key the same way as the authenticator. In the second part of the application, the RP is provided with the credential identifier and the shared key of an authenticator. The RP creates an encoded COSE Encrypt structure that contains an extension value encrypted with a content encryption key. This key is then encrypted using the shared key and attached as a recipient object. The credential id of the corresponding authenticator is used as

[2] https://github.com/Digital-Security-Lab/protecting-fido-extensions-poc.
[3] https://github.com/abuettner/cose-lib.

key identifier. The authenticator receives this COSE Encrypt structure and identifies that a recipient is attached with its credential id. It can then decrypt the content encryption key and finally the extension value. The authenticator then creates an encoded COSE Encrypt0 structure that contains another extension value, this time encrypted with the shared key. The RP receives this structure and finally decrypts the extension value by the authenticator. Note that in a real application, the RP can identify the shared key used by the credential id that is part of the authenticator data.

Preliminary measurements on a Raspberry Pi Pico (264 kB SRAM, 2 MB on-board flash memory) [31] show that the steps performed by an authenticator during the registration take about 250 ms, while the steps during the assertion take about 5 ms. The additional delay during registration is acceptable, since this is performed only once per application. The additional runtime on assertions would be unnoticeable by the user.

8 Discussion

In this section we discuss our proposed protocol for protecting FIDO extensions with regard to several different aspects.

8.1 Security

There are several different ways to intercept FIDO messages in clear text as described in Sect. 4. This allows an attacker to intercept FIDO extensions with valuable information and either eavesdrop or manipulate them. As shown by the evaluation the security of FIDO extensions can be significantly improved with our proposed solution. However, the security of extensions also depends on a secure key exchange. While RPs can verify the attestation to get information on security properties provided by an authenticator to create trust, authenticators cannot reliably verify the origin of a RP. In our formal models, the client between the RP and the authenticator has not been considered. It could be argued that the client adds security to some extent, e.g., by validating the TLS certificate of the server. Yet, this is not sufficient and additional measures as proposed in this paper are justified.

The security also depends on the strength of cryptographic algorithms. This has not been evaluated within this work. We consider cryptographic algorithms that are widely accepted and used e.g. in the most recent TLS 1.3 [32]. However, the proposed protocol is meant to be generic, so cryptographic algorithms can simply be replaced if necessary (e.g. with post-quantum cryptography).

8.2 Implementation

We provide an example on how our protocol can be implemented. Our protocol is completely compatible with the FIDO standards. While the protocol is quite complex, our implementation can be used to integrate it into FIDO applications

with low effort. Since there are not too many COSE library implementations, a further contribution of this work is such a COSE library which can be used by any other C application.

At the time of writing, FIDO implementations are quite restricted to standardized extensions. Even though the WebAuthn standard [22] defines how arbitrary extensions should be forwarded to the authenticator, browsers have not implemented this. This means that custom extensions are not passed to FIDO devices. It is therefore challenging to implement a real-world example at this stage. Our protocol should also be considered for extensions that will be standardized in the future, such as, the Secure Payment Confirmation [27] which is clearly an extension with high security requirements.

8.3 Usability

Usability is an important aspect that can affect the user experience and acceptance. It is an essential criterion that will certainly determine how successful FIDO authentication will become in the future. The usability for FIDO authentication is, however, not affected by our proposed protocol. The protocol requires a key exchange and subsequent encryption of FIDO extensions, which happens autonomously and is therefore unnoticed by the user.

9 Conclusion and Outlook

The FIDO protocols are a promising way to prevent security risks that arise with password authentication. However, we describe several MitM attacks which show that FIDO extensions are vulnerable to disclosure and manipulation. In order to mitigate such attacks, we propose a protocol that secures FIDO extensions by authenticated encryption. While our methodology includes some challenges such as the initial key exchange and displaying user information for authenticators without a secure display, we see a considerable security gain and aim for a standardized way to secure any kind of FIDO extension.

There are not many extensions used in practice yet. Nevertheless, the standardization process of the Secure Payment Confirmation indicates that we can expect more extensions to appear in the near future. At the time of writing, it is still under discussion if arbitrary extensions should be allowed or not. We argue that it would be beneficial from a developer's perspective to be able to add extensions for different applications. This should, however, be done with security in mind. The protocol presented in this paper could provide a way to satisfy this requirement. In future work, we will test our approach in real world scenarios. Furthermore, we are working on a lab environment that will facilitate practical research with FIDO authentication.

References

1. Akter, S., Chellappan, S., Chakraborty, T., Khan, T.A., Rahman, A., Al Islam, A.A.: Man-in-the-middle attack on contactless payment over NFC communications: design implementation, experiments and detection. IEEE Trans. Depend. Secur. Comput. **18**, 3012–3023 (2020)
2. Arshad, S., Kharraz, A., Robertson, W.: Include me out: in-browser detection of malicious third-party content inclusions. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 441–459. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_26
3. Barbosa, M., Boldyreva, A., Chen, S., Warinschi, B.: Provable security analysis of FIDO2. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 125–156. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_5
4. Bianchi, A., Corbetta, J., Invernizzi, L., Fratantonio, Y., Kruegel, C., Vigna, G.: What the app is that? Deception and countermeasures in the android user interface. In: 2015 IEEE Symposium on Security and Privacy, pp. 931–948. IEEE (2015)
5. Blanchet, B.: Modeling and verifying security protocols with the applied pi calculus and ProVerif. Found. Trends® Priv. Secur. **1**(1–2), 1–135 (2016)
6. Bormann, C., Hoffman, P.E.: Concise Binary Object Representation (CBOR). RFC 8949, December 2020. https://doi.org/10.17487/RFC8949, https://rfc-editor.org/rfc/rfc8949.txt
7. Bui, T., Rao, S.P., Antikainen, M., Bojan, V.M., Aura, T.: Man-in-the-machine: exploiting ill-secured communication inside the computer. In: 27th USENIX Security Symposium (USENIX Security 2018), pp. 1511–1525 (2018)
8. Büttner, A., Gruschka, N.: Enhancing FIDO Transaction Confirmation with Structured Data Formats. In: Norsk IKT-konferanse for forskning og utdanning. No. 3 (2021)
9. Büttner, A., Nguyen, H.V., Gruschka, N., Lo Iacono, L.: Less is often more: header whitelisting as semantic gap mitigation in HTTP-based software systems. In: Jøsang, A., Futcher, L., Hagen, J. (eds.) SEC 2021. IAICT, vol. 625, pp. 332–347. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78120-0_22
10. Dougan, T., Curran, K.: Man in the browser attacks. Int. J. Amb. Comput. Intell. (IJACI) **4**(1), 29–39 (2012)
11. Feng, H., Li, H., Pan, X., Zhao, Z.: A formal analysis of the FIDO UAF protocol. In: Proceedings of 28th Network And Distributed System Security Symposium (NDSS) (2021)
12. Fernandes, E., et al.: Android UI deception revisited: attacks and defenses. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 41–59. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_3
13. FIDO Alliance: FIDO Transaction Confirmation White Paper. Technical report, August 2020. https://media.fidoalliance.org/wp-content/uploads/2020/08/FIDO-Alliance-Transaction-Confirmation-White-Paper-08-18-DM.pdf
14. FIDO Alliance: Fido alliance metadata service (2021). https://fidoalliance.org/metadata/
15. FIDO Alliance: Fido alliance specifications overview (2021). https://fidoalliance.org/specifications/
16. FIDO Alliance: History of fido alliance (2021). https://fidoalliance.org/overview/history/

17. Frymann, N., Gardham, D., Kiefer, F., Lundberg, E., Manulis, M., Nilsson, D.: Asynchronous remote key generation: an analysis of Yubico's proposal for W3C WebAuthn. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 939–954 (2020)
18. Gil, O.: Web cache deception attack. Black Hat USA 2017 (2017)
19. Google: Fido2 API for android (2020). https://developers.google.com/identity/fido/android/native-apps
20. Group, W.W.A.W.: Web authentication (webauthn) (2020). https://www.iana.org/assignments/webauthn/webauthn.xhtml
21. Jakkal, V.: The passwordless future is here for your microsoft account (2021). https://www.microsoft.com/security/blog/2021/09/15/the-passwordless-future-is-here-for-your-microsoft-account/
22. Kumar, A., Jones, J., Hodges, J., Jones, M., Lundberg, E.: Web authentication: an API for accessing public key credentials - level 2. In: W3C recommendation, W3C, April 2021. https://www.w3.org/TR/2021/REC-webauthn-2-20210408/
23. Kunke, J., Wiefling, S., Ullmann, M., Lo Iacono, L.: Evaluation of account recovery strategies with fido2-based passwordless authentication. In: Roßnagel, H., Schunck, C.H., Mödersheim, S. (eds.) Open Identity Summit 2021, pp. 59–70. Gesellschaft für Informatik e.V, Bonn (2021)
24. Lahmadi, A., Duque, A., Heraief, N., Francq, J.: MitM attack detection in BLE networks using reconstruction and classification machine learning techniques. In: Koprinska, I., et al. (eds.) ECML PKDD 2020. CCIS, vol. 1323, pp. 149–164. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65965-3_10
25. Landrock, P., Pedersen, T.: WYSIWYS?-What you see is what you sign? Inf. Secur. Techn. Rep. **3**(2), 55–61 (1998)
26. Linhart, C., Klein, A., Heled, R., Steve, O.: HTTP Request Smuggling (2005). https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf
27. McGruer, S., Solomakhin, R.: Secure Payment Confirmation. In: W3C working draft, W3C, August 2021. https://www.w3.org/TR/2021/WD-secure-payment-confirmation-20210831/
28. Owens, K., Anise, O., Krauss, A., Ur, B.: user perceptions of the usability and security of smartphones as FIDO2 roaming authenticators. In: Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021), pp. 57–76 (2021)
29. Pfeffer, K., et al.: On the usability of authenticity checks for hardware security tokens. In: 30th USENIX Security Symposium (USENIX Security 2021) (2021)
30. Porter, J.: Safari to support password-less logins via face id and touch id later this year (2020). https://www.theverge.com/2020/6/24/21301509/apple-safari-14-browser-face-touch-id-logins-webauthn-fido2
31. Raspberry Pi Ltd: Raspberry Pi Documentation - Raspberry Pi Pico (2022). https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html
32. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. https://doi.org/10.17487/RFC8446, https://rfc-editor.org/rfc/rfc8446.txt
33. Schaad, J.: CBOR Object Signing and Encryption (COSE). RFC 8152, July 2017. https://doi.org/10.17487/RFC8152, https://rfc-editor.org/rfc/rfc8152.txt
34. Selander, G., Mattsson, J.P., Palombini, F.: Ephemeral Diffie-Hellman Over COSE (EDHOC). Internet-Draft draft-ietf-lake-edhoc-12, Internet Engineering Task Force, October 2021. https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-12. (work in Progress)

35. Sun, D.Z., Mu, Y., Susilo, W.: Man-in-the-middle attacks on secure simple pairing in Bluetooth standard V5. 0 and its countermeasure. Pers. Ubiquit. Comput. **22**(1), 55–67 (2018)
36. Wendlandt, D., Andersen, D.G., Perrig, A.: Perspectives: improving ssh-style host authentication with multi-path probing. In: USENIX Annual Technical Conference, vol. 8, pp. 321–334 (2008)
37. Xu, P., Sun, R., Wang, W., Chen, T., Zheng, Y., Jin, H.: SDD: a trusted display of FIDO2 transaction confirmation without trusted execution environment. Future Gener. Comput. Syst. **125**, 32–40 (2021)
38. Zhang, Y., Wang, X., Zhao, Z., Li, H.: Secure display for FIDO transaction confirmation. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, pp. 155–157 (2018)
39. Zhang, Z., Diao, W., Hu, C., Guo, S., Zuo, C., Li, L.: An empirical study of potentially malicious third-party libraries in Android apps. In: Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 144–154 (2020)