Credential Exchange Protocol

Working Draft, May 22, 2024

This version:
https://fidoalliance.org/specs/cx/cxp-v1.0-wd-20240522.html

Issue Tracking:
Github
GitHub

Editor:
Nick Steele (1Password)

Contributors:
Rew Islam (Dashlane)
Anders Åberg (Bitwarden)
René Léveillé (1Password)
Oscar Hinton (Bitwarden)
Jonathan Salamon (Dashlane)
Ayman Bedair (NordPass)
Lee Campbell (Google)
Reema Bajwa (Google)

Abstract

This document defines a protocol to securely move one or more credentials between two credential providing applications same or separate devices.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current FIDO Alliance publications and the latest revision of this technical report can be found in the FIDO Alliance specifications index at https://fidoalliance.org/specifications/.

This document was published by the FIDO Alliance as a Working Draft Specification. If you wish to make comments regarding this document, please Contact Us. All comments are welcome.

This is a Working Draft Specification and is not intended to be a basis for any implementations as the Specification may change. This document is merely a FIDO Alliance working group internal and **member-confidential** document. It has no official standing of any kind and does not represent consensus of the FIDO Alliance. No rights are granted to prepare derivative works of this Specification. Entities seeking permission to reproduce portions of this Specification for other uses must contact the FIDO Alliance to determine whether an appropriate license for such use is available.

Table of Contents

1. Introduction§

NOTE: The name of this specification is subject to change.

Individuals and organizations use credential providers to create and manage credentials on their behalf as a means to use stronger authentication factors. These credential providers can be used in browsers, on network servers, and on mobile and desktop platforms, and often sharing or synchronizing credentials between different instances of the same provider is an easy and common task.

However, the transfer of credentials between two different providers has traditionally been an infrequent occurrence, such as when a user or organization is attempting to migrate credentials from one provider to another. As it becomes more common for users to have multiple credential providers that they use to create a manage credentials, it becomes important to address some of the security concerns with regard to migration currently:

- Credential provider applications often export credentials to be imported in an insecure format, such as CSV, that undermines the security of the provider and potentially opens the credential owner to vulnerability.
- Credential providers have no standard structure for the exported credential CSV, which can sometimes result in failure to properly migrate one or more credentials into a new provider.
- Some credentials might be unallowed to be migrated, due to device policy or lack of algorithmic capability by the importing credential provider.
- Because organizations lack a secure means of migrating user credentials, often they will apply device policy that prevents the export of credentials to a new provider under any circumstances, opting to create multiple credentials for a service.

In order to support credential provider interoperability and provide a more secure means of credential transfer between providers, this document outlines a protocol for the import and export of one or more credentials between two credential providers on behalf of a user or organization in both an offline or online context. Using Diffie-Hellman key exchange, this protocol allows the creation of a secure channel or data payload between two providers.

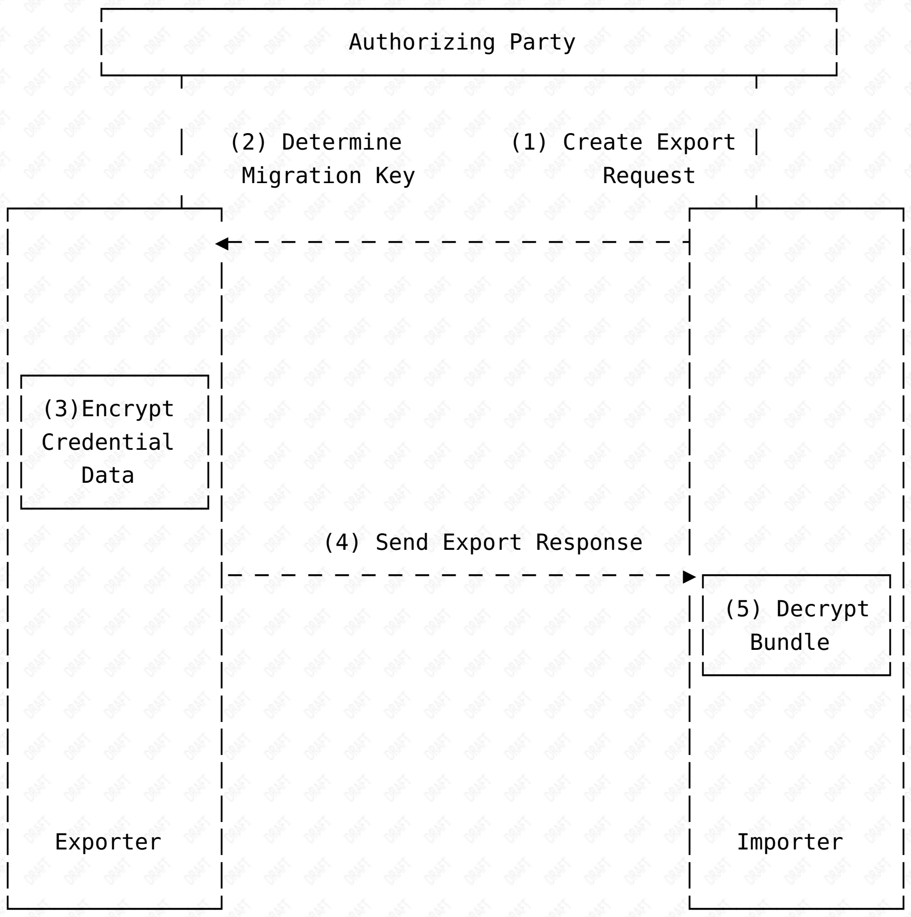
1.1. Scope§

This protocol describes the secure transmission of one or more credentials between two credential providers on the same or different devices managed by the same credential owner, capable of function in both online and offline contexts. This protocol does not make any assumptions about the channels in which credential data is passed from the source provider to the destination provider. The destruction of credentials after migration by the credential provider source is out of scope as well.

1.2. Terminology§

{::boilerplate bcp14-tagged} Certain security-related terms are to be understood in the sense defined in[RFC4949]. These terms include, but are not limited to, "attack", "authentication" "authorization", "certificate", "credential", "encryption", "identity", "sign", "signature", "trust", "validate", and "verify".

2. Protocol Overview§



Authorizing Party
(2) Determine Migration Key
(1) Create Export Request
(3)Encrypt Credential Data
(4) Send Export Response
(5) Decrypt Bundle
Exporter
Importer

The flow illustrated above shows the following:

1. The importing credential provider initiates the flow by creating an export request for the exporting provider. The import request includes a challenge, the scope of the export request, and a declaration of the type of encryption scheme to be used. In most cases, this will use Diffie-Hellman and the importer will provide a public key in the export request.
2. If an end-user and/or authorizing party approves the request, the exporter uses the export request key to generate or retrieve the migration key used to encrypt the credential data.
3. The source collects and encrypts the requested credential data for export and signs the challenge provided by the importer.
4. The export response is sent to the destination provider and includes the encrypted credential data, the signed challenge response, and the public key of the source credential provider’s migration key.
5. The importing provider validates the challenge and retreives the migration key, decrypting the credential data which is formatted normatively with (CXF), and then stores the result.

2.1. Participants§

2.1.1. Credential Owner§

An entity that is able to authenticate, authorize, or access one or more credentials from within a credential provider. The credential owner is in charge of authorizing or delegating authorization of the migration between the source credential provider and the destination provider. In the case of a credential owner being an individual, they are referred to as the end-user, an individual or service being delegated by an organization are considered credential owner agents.

2.1.2. Credential Providers§

Hardware or software capable of securely storing and managing credentials on behalf of their owner. The two credential providers that participate in this protocol should generally be two distinct pieces of hardware or software.

*2.1.2.1. **Importing Provider**§*

The importing provider, or importer, initiates the export request and is the final storage destination for the exported credentials.

*2.1.2.2. **Exporting Provider**§*

The exporting provider, or exporter, encrypts and transfers the credential data to the importing provider.

2.1.3. Authorizing Party§

An OPTIONAL authority that can grant and attest certificates on behalf of a credential owner. These certificates are used to authenticate the credential agent and MAY be used to create the migration key used on behalf of the source and destination credential providers.

2.2. Supporting Different Network Conditions§

This protocol can work in both online and offline scenarios, as well as in air-gapped networks where one or both devices might not have access to the internet. Different network conditions might result in participants like an authorizing party or another outside service from being included, but the core exchange protocol should not be affected.

2.3. Supporting Different Key Encryption Schemes§

Import and export providers can use different encryption schemes to initiate a secure exchange of credentials, and depending on the needs of the credential owner, there may be a preference for which type of scheme is used for encrypting the credential payload.

2.3.1. Diffie-Hellman Key Exchange {#dhke}§

This scheme allows for an importer and exporter to negotiate a shared key using a key pair that SHOULD be generated by the respective provider applications for the purposes of migration, but MAY be supplied by an authorizing party in enterprise cases.

2.3.2. Hybrid Public Key Encryption {#hpke}§

If the importing provider receives or knows the exporter’s public key prior to generating the export request, it is capable of hybrid encryption to encapsulate a migration key that can be securely passed to the exporting application.

3. Protocol API§

3.1. Credential Types§

Credential Types are defined through the Credential Exchange Format (CXF). The exported credentials MUST be formatted using CXF in order to have interoperability.

3.2. Export Request§

The export request initiates the protocol which contains a set of encryption parameters. These encryption parameters MUST have an associated public key if it is necessary for that instance of given parameters. This request or elements of it MAY be created by the authorizing party if one is present, or created by the destination credential provider with or without input from the credential owner.

```
dictionary ExportRequest {
    required unsigned short version = 0;
    required sequence<HPKEParameters> hpke;
    required sequence<DOMString> archive;
    required DOMString importer;
    sequence<DOMString> credentialTypes;
    sequence<DOMString> knownExtensions;
  };
```

`version`, of type unsigned short, defaulting to `0`
: The protocol version that the `Importing Provider` wants to use in the exchange. This document currently details version 0.

`hpke`, of type sequence<HPKEParameters>
: This member defines a list of `HPKEParameters` that the `Importing Provider` supports in order of preference. It is up to the `Exporting Provider` to select a matching set of parameters that both support.

`archive`, of type sequence<DOMString>
: This member defines a list of archiving algorithms that the `Importing Provider` supports in order of preference. It is up to the `Exporting Provider` to select an algorithm that both support. The values of this list SHOULD be members of `ArchiveAlgorithm` and the `Exporting Provider` MUST ignore any unknown values.

`importer`, of type DOMString
: This member is the `Relying Party Identifier` of the `Importing Provider`.

`credentialTypes`, of type sequence<DOMString>
: This OPTIONAL member lists the types of credentials that the `Importing Provider` understands and requests from the `Exporting Provider`. This list SHOULD be validated by the user before initiating the exchange. The values in the list SHOULD be members of `CredentialType` and the `Exporting Provider` MUST ignore any unknown values.

  If this member is not present then it is understood that the `Importing Provider` is requesting all credential types. If this member is present but the list is empty, the `Exporting Provider` MUST send only the `Account` object without any `Collection` information.

`knownExtensions`, of type sequence<DOMString>
: This OPTIONAL member lists the extensions that the `Importing Provider` understands. This list SHOULD be members of name defined in CXF and the `Exporting Provider` MUST ignore all unknown values.

  If this member is not present, then it is understood that the `Importing Provider` is requesting all extensions that the `Exporting Provider` wishes to include. If this member is present but the list is empty, the `Exporting Provider` MUST NOT include any extensions in the resulting export.

3.3. Export Response§

The export response includes both the credential payload and any metadata necessary to decrypt and marshall the credentials into the importer’s storage.

```
dictionary ExportResponse {
    required unsigned short version = 0;
    required HPKEParameters hpke;
    required DOMString archive;
    required DOMString exporter;
    required Base64UrlString payload;
};
```

`version`, of type unsigned short, defaulting to `0`
: The protocol version that the `Exporting Provider` understands. The value SHOULD be the same as `version` however there is the possibility of the`Exporting Provider` having a previous version of the protocol implemented and therefore responding with a lower version. The `Importing Provider` MAY refuse this version downgrade. This document currently details version 0.

`hpke`, of type HPKEParameters
: This member defines the encryption parameters selected by the`Exporting Provider`. The value MUST correspond to an entry in `hpke`.

`archive`, of type DOMString
: This member defines the archiving algorithm selected by the`Exporting Provider`. The value MUST correspond to an entry in `archive`.

`exporter`, of type DOMString
: This member is the`Relying Party Identifier` or the`Exporting Provider`.

`payload`, of type Base64UrlString
: This contains the base64url encoded`Credential Payload`.

3.4. Credential Payload§

One or more normatively formatted credentials that are passed inside the export response. The format MUST follow the zip archive format as defined in (CXF) where each file is separately encrypted using the key defined by the selected `HPKEParameters`. The file names are replaced with the anonymous identifier in the export request. All files are stored as JSON Web Encryption files.

```
CXP-Export/
├─ index.jwe
├─ documents/
|  ├─ 1b3.jwe
|  ├─ d5f.jwe
|  ├─ 7h9.jwe
```

3.5. Supporting Types§

3.5.1. HPKE Parameters§

```
dictionary HPKEParameters {
    required DOMString mode;
    required unsigned short kem;
    required unsigned short kdf;
    required unsigned short aead;
    JWK key;
};
```

`mode`, of type DOMString
: The encryption mode as defined in[RFC9180] and SHOULD be a member of`HPKEMode`. The `Exporting Provider` SHOULD ignore any `HPKEParameters` where this value is unknown.

`kem`, of type unsigned short
: The encryption key encapsulation method as defined in[RFC9180]. The value SHOULD be from the HPKE

KEM Identifiers IANA table, the `Exporting Provider` SHOULD ignore any `HPKEParameters` where this value is unknown.

`kdf`, of type unsigned short

The encryption key derivation function as defined in[RFC9180]. The value SHOULD be from the HPKE KDF Identifiers IANA table, the `Exporting Provider` SHOULD ignore any `HPKEParameters` where this value is unknown.

aead, of type unsigned short

The authenticated encryption with additional data algorithm as defined in[RFC9180]. The value SHOULD be from the HPKE AEAD Identifiers IANA table, the `Exporting Provider` SHOULD ignore any `HPKEParameters` where this value is unknown.

key, of type JWK

This member is only present in the case that the option is not using a pre-shared key. It is a JSON Web Key representing that provider's key necessary for the other provider's generation of the AEAD key.

3.5.2. HPKE Modes§

```
enum HPKEMode{
    "base",
    "psk",
    "auth",
    "auth-psk"
};
```

`base`

Base mode of encrypting a public key.

`psk`

Authenticates the possession of a pre-shared key.

`auth`

Authenticates the possession of a KEM private key.

`auth-psk`

Authenticates possession of both a pre-shared key and a KEM private key.

3.5.3. Archive Algorithms§

```
enum ArchiveAlgorithm {
    "deflate"
};
```

`deflate`

Archiving through the use of the DEFLATE algorithm defined in[RFC1951].

4. IANA Considerations§

This document has no IANA actions.

5. Implementation Requirements§

This section defines which algorithms and features of this specification are mandatory to implement. Applications using this specification can impose additional requirements upon implementations that they use.

6. Security Considerations§

TODO Security

Conformance§

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

> EXAMPLE 1 ¶
> This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

> Note, this is an informative note.

Index§

Terms defined by this specification§

Terms defined by reference§

[WEBIDL] defines the following terms:

- DOMString
- sequence
- unsigned short

References§

Normative References§

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[WEBIDL]
Edgar Chen; Timothy Gu. *Web IDL Standard*. Living Standard. URL: https://webidl.spec.whatwg.org/

Informative References§

[RFC1951]
P. Deutsch. *DEFLATE Compressed Data Format Specification version 1.3*. May 1996. Informational. URL: https://www.rfc-editor.org/rfc/rfc1951

[RFC4949]
R. Shirey. *Internet Security Glossary, Version 2*. August 2007. Informational. URL: https://www.rfc-editor.org/rfc/rfc4949

[RFC9180]
R. Barnes; et al. *Hybrid Public Key Encryption*. February 2022. Informational. URL: https://www.rfc-editor.org/rfc/rfc9180

IDL Index§

```
dictionary ExportRequest {
    required unsigned short version = 0;
    required sequence<HPKEParameters> hpke;
    required sequence<DOMString> archive;
    required DOMString importer;
    sequence<DOMString> credentialTypes;
    sequence<DOMString> knownExtensions;
  };

dictionary ExportResponse {
    required unsigned short version = 0;
    required HPKEParameters hpke;
    required DOMString archive;
    required DOMString exporter;
    required Base64UrlString payload;
};

dictionary HPKEParameters {
    required DOMString mode;
    required unsigned short kem;
    required unsigned short kdf;
    required unsigned short aead;
    JWK key;
};

enum HPKEMode{
    "base",
    "psk",
    "auth",
    "auth-psk"
};

enum ArchiveAlgorithm {
    "deflate"
};
```