Credential Exchange Format

Working Draft, October 03, 2024

This version:
https://fidoalliance.org/specs/cx/cxf-v1.0-wd-20241003.html

Issue Tracking:
GitHub
GitHub

Editor:
René Léveillé (1Password)

Contributors:
Nick Steele (1Password)
Rew Islam (Dashlane)
Anders Åberg (Bitwarden)
Oscar Hinton (Bitwarden)
Jonathan Salamon (Dashlane)
Ayman Bedair (NordPass)
Lee Campbell (Google)
Reema Bajwa (Google)
Hans Reichenbach (Okta)

Abstract

This document defines the data structures and format of credentials being passed or referenced between two applications during credential exchange.

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.

Table of Contents

1. Introduction§

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

Credential migration has traditionally been an infrequent occurrence, when a user is attempting to migrate credentials from one credential provider to a new one, such as moving to a new password manager or mobile device. This has historically been a very manual process for credential providers, as there exists no normative structure to the credentials being exported by a credential provider. The goal of CXF is to define those normative data structures to allow for interoperability and control by resource owners over credentials that need to be migrated or referenced by one or more providers.

1.1. Motivation§

Historically, there is no normative structure for passing credentials between credential providers, leading to a lack of interoperability and in some cases, the loss of credentials during transfer. While the Credential Exchange Protocol aims to define the standard protocol for the import and export of credentials, there additionally needs to be a standard format for the credential data being exchanged. The Credential Exchange Format aims to solve non-normative credential transfer for this protocol and other forms of credential exchange between providers to help make the process easier for users and organizations to securely handle exchange events.

1.2. Scope§

This document outlines the data structures and format needed to exchange credentials and does not make any assumptions about the protocol used for the transfer, such as the protocol outlined by CXP.

1.3. Terminology§

[Define any key terms and concepts used throughout this document.]

2. Format Overview§

CXF defines a schema around an account owner and all of its associated secrets. These secrets are defined in a way where the most common attributes have dedicated fields, all the while allowing extra fields to be added as extensions.

2.1. Format Design Principles§

Everything in a zip archive, each part is encrypted using the keys defined inCXP.

```
CXF-Export/
├─ index.json
├─ documents/
│  ├─ foo.docx
│  ├─ vault.ico
│  ├─ bar.pdf
```

[Detail the key principles that guided the design of the CXF format.]

2.2. Data Structures§

[Explain the overall structure of CXF, including its main sections and their purposes.]

2.3. Encoding Considerations§

[Discuss considerations related to encoding and data representation within the CXF format.]

3. Data Structure Specification§

[Provide detailed specifications for each section of the CXF data structure.]

3.1. Header Section§

[Describe the contents and purpose of the header section within the CXF data structure.]

3.1.1. Header§

```
Header = {
    version: uint .size 2,
    exporter: tstr,
    timestamp: uint .size 8,
    accounts: [ * Account ],
}
```

version
The version of the format definition, The current version is `0`.

exporter
The name of the exporting app as a relying party identifier.

timestamp
The UNIX timestamp during at which the export document was completed.

accounts
The list of Accounts being exported.

3.1.2. Account Dictionary§

```
Account = {
    id: b64url,
    userName: tstr,
    email: tstr,
    ? fullName: tstr,
    ? icon: tstr,
    collections: [ * Collection ],
    items: [ * Item ],
    ? extensions: [ * Extension ] .default [],
}
```

id
: A unique identifier for the Account which is machine generated and an opaque byte sequence with a maximum size of 64 bytes. It SHOULD NOT to be displayed to the user.

userName
: A pseudonym defined by the user to name their account. If none is set, this should be an empty string.

email
: The email used to register the account in the previous provider.

fullName
: This OPTIONAL field holds the user’s full name.

icon
: This OPTIONAL field defines if the user has set an icon as the account’s avatar.

collections
: All the collections this account owns. If the user has collections that were shared with them by another account, it MUST NOT be present in this list.

items
: All items that this account owns. If the user has access to items that were shared with them by another account, it MUST NOT be present in this list.

extensions
: This OPTIONAL field contains all the extensions to theAccount’s attributes.

3.1.3. Collection Dictionary§

```
Collection = {
    id: b64url,
    title: tstr,
    ? subtitle: tstr,
    ? icon: tstr,
    items: [ * LinkedItem ],
    ? subCollections: [ * Collection ] .default [],
    ? extensions: [ * Extension ] .default [],
}
```

id
: A unique identifier for the Collection which is machine generated and an opaque byte sequence with a maximum size of 64 bytes. It SHOULD NOT be displayed to the user.

title
: The display name of the Collection.

subtitle
: This OPTIONAL field is a subtitle or a description of theCollection.

icon
: This OPTIONAL field is a relative path from this file to the icon file acting as thisCollection’s avatar.

items

Enumerates all the LinkedItem in this Collection. A LinkedItem contains the necessary data to indicate which Items are part of this Collection.

subCollections
Enumerates any sub-collections if the provider supports recursive organization.

extensions
This enumeration contains all the extensions to the Collection's attributes.

3.2. Credential Section§

[Explain the components and fields of the credential section, detailing how credentials are represented.]

3.2.1. Item Dictionary§

```
Item = {
    id: b64url,
    creationAt: uint,
    modifiedAt: uint,
    type: ItemType / tstr,
    title: tstr,
    ? subtitle: tstr,
    credentials: [ * Credential ],
    ? tags: [ * tstr ] .default [],
    ? extensions: [ * Extension ] .default [],
}
```

id
A unique identifier for the Item which is machine generated and an opaque byte sequence with a maximum size of 64 bytes. It SHOULD NOT be displayed to the user.

creationAt
The UNIX timestamp at which this item was originally created.

modifiedAt
The UNIX timestamp of the last modification brought to this Item.

type
This member contains a hint to the objects in the credentials array. It SHOULD be a member of ItemType.

title
This member's value is the user-defined name or title of the item.

subtitle
This OPTIONAL member is a subtitle or description for the Item.

credentials
This member contains a set of Credentials that SHOULD be associated to the type.

tags
This OPTIONAL member contains user-defined tags that they may use to organize the item.

extensions
This member contains all the extensions the exporter MAY have to define the Item type that is being exported to be as complete of an export as possible.

3.2.2. LinkedItem Dictionary§

```
LinkedItem = {
    item: b64url,
    ? account: b64url,
}
```

item

The Item’s id that this LinkedItem refers to. Note that this Item might not be sent as part of the current exchange.

account

This OPTIONAL member indicates the Account’s id the referenced Item belongs to. If not present, the Item belongs to the current Account being exchanged.

3.3. Credential Data Types§

3.3.1. Credential Base Dictionary§

```
Credential = $Credential .within {
    type: CredentialType / tstr
}
```

type

This member contains a **string representation of the credential type**. The value SHOULD be a member of CredentialType but importers MAY attempt to store unknown item types in their own way as a best effort.

> NOTE: The type value will be the same for all items implementing a particular credential which means that developers can rely on `obj.type` returning a string that unambiguously represents the specific kind of Credential they are dealing with.

3.3.2. BasicAuth§

```
$Credential /= BasicAuth
BasicAuth = {
    type: "basic-auth",
    urls: [ * uri ],
    ? username: EditableField,
    ? password: EditableField
}
```

3.3.3. Passkey Dictionary§

```
$Credential /= Passkey
Passkey = {
    type: "passkey",
    credentialId: b64url,
    rpId: tstr,
    userName: tstr,
    userDisplayName: tstr,
    userHandle: b64url,
    key: b64url,
    ? fido2Extensions: Fido2Extensions,
}
```

type
: This overriden member from Credential MUST be present and MUST have a value ofpasskey.

credentialId
: This member contains a WebAuthn Credential ID which uniquely identifies the passkey instance. The decoded raw value MUST be equal to the value given in `PublicKeyCredential`'s `rawId` field during registration.

rpId
: This member specifies the WebAuthn Relying Party Identifier to which the passkey instance is tied to. The value MUST be equal to the RP ID that was defined by the authenticator during credentialregistration.

userName
: This member contains a human-palatable identifier for the user account to which the passkey instance is tied to. The value SHOULD be equal to the value in `PublicKeyCredentialUserEntity`'s `name` member given to the authenticator during registration.

  The only case where the value MAY not be the one set duringregistration is if the exporting provider allows the user to edit their username. In such a case, the value of this field MUST be the user edited value. See § 3.3.3.1 Editability of passkey fields for more details.

userDisplayName
: This member represents a human-palatable name for the user account, intended only for display. The value SHOULD be equal to the value in `PublicKeyCredentialUserEntity`'s `displayName` member given to the authenticator during registration.

  The only case where the value MAY not be the one set duringregistration is if the exporting provider allows the user to edit their user display name. In such a case, the value of this field MUST be the user edited value. See § 3.3.3.1 Editability of passkey fields for more details.

userHandle
: This member contains the user handle which is the value used to identify theuser account associated to this passkey instance. The value MUST be equal to the value in `PublicKeyCredentialUserEntity`'s `id` member given to the authenticator during registration

key
: The private key associated to this passkey instance. The value MUST bePKCS#8 formatted byte string which is then [[!RFC4648#section-5|Base64url encoded]]. The value MUST give the same public key value that was provided by the original authenticator during registration.

fido2Extensions
: This OPTIONAL member denotes the WebAuthn or CTAP2 extensions that are associated to this passkey instance.

NOTE: Passkeys using a non-zero signature counter MUST be excluded from the export and the exporter SHOULD inform the user that such passkeys are excluded from the export. Importers MUST set a zero value for the imported passkey signature counters and MUST NOT increment them after the fact.

3.3.3.1. Editability of passkey fields§

Note that there are certain members of the Passkey dictionary that are marked as being editable by the user. Only human-palatable values MAY be edited by the user since these are not REQUIRED forWebAuthn ceremonies. These member also represent values that MAY be changed by the user on the relying party. Exporting providers MAY let users to edit these members to mirror the changes on therelying party. In such cases the value at the time of exchange MUST be the user edited value. The only accepted user editable Passkey fields are:

- userName
- userDisplayName

All other members of the Passkey dictionary MUST NOT be user editable as they are required for the WebAuthn ceremonies to be successful.

3.3.4. CreditCard§

```
$Credential /= CreditCard
CreditCard = {
    type: "credit-card",
    number: tstr,
    fullName: tstr,
    ? cardType: tstr,
    ? verificationNumber: tstr,
    ? expiryDate: tstr,
    ? validFrom: tstr,
}
```

3.3.5. `Note`§

```
$Credential /= Note
Note = {
    type: "note",
    content: tstr,
};
```

type
: This overriden member from Credential MUST be present and MUST have a value ofnote.

content
: This member is a user-defined value encoded as a UTF-8 string.

3.3.6. TOTP§

NOTE: Enrollment in TOTP credentials historically has been quite non-standardized but typically authenticator and RP implementations have more or less aligned with the early Google Authenticator implementation spelled out at https://github.com/google/google-authenticator/wiki/Key-Uri-Format. This specification was designed with that in mind.

```
$Credential /= TOTP
TOTP = {
    type: "totp",
    secret: tstr,
    period: uint .size 2,
    digits: uint .size 2,
    username: tstr,
    algorithm: OTPHashAlgorithm / tstr,
    ? issuer: tstr,
}
```

secret
: The [[!RFC4226#section-4|shared secret]] used to generate the OTPs. This MUST be a [[!RFC4648#section-6|Base32 string]]

period
: The time step used to refresh the OTP in seconds. The default SHOULD be 30 seconds, although the relying party MAY customize this to a different value.

digits

The number of digits to generate and display to the user each period. The default SHOULD be 6, although the relying party MAY customize this to a different value.

username

The username of the account this TOTP credential is used for.

algorithm

The algorithm used to generate the OTP hashes. This value SHOULD be a member of OTPHashAlgorithm but importers MUST ignore TOTP entries with unknown algorithm values.

issuer

This OPTIONAL member contains the relying party that issued the credential and should be user consumable.

> NOTE: While this member is optional, it is strongly recommended to be included if available.

3.4. Metadata Section§

[Detail the metadata section’s role in providing additional information about the credential data.]

3.5. Supporting Data Structures§

3.5.1. ItemType Enumeration§

```
ItemType =
    "login" /
    "document" /
    "identity"
```

login

An Item that SHOULD contain any of the following Credential types:

- BasicAuth,
- Passkey,
- TOTP,
- CryptographicKey.

document

An Item that SHOULD contain any of the following Credential types:

- Note,
- File.

identity

An Item that SHOULD contain any of the following Credential types:

- CreditCard
- Address
- DriverLicense
- SocialSecurityNumber

3.5.2. CredentialType Enumeration§

```
CredentialType =
    "basic-auth" /
    "passkey" /
    "totp" /
    "cryptographic-key" /
    "note" /
    "file" /
    "address" /
    "credit-card" /
    "social-security-number"
```

basic-auth

passkey

totp

cryptographic-key

note

file

address

credit-card

social-security-number

3.5.3. OTPHashAlgorithm Enumeration§

```
OTPHashAlgorithm =
    "sha1" /
    "sha256" /
    "sha512"
```

3.5.4. EditableField Dictionary§

```
EditableField = {
    id: b64url,
    fieldType: FieldType / tstr,
    value: tstr,
    ? label: tstr
}
```

id

A unique identifier for the EditableField which is machine generated and an opaque byte sequence with a maximum size of 64 bytes. It SHOULD NOT be displayed to the user.

fieldType

This member defines the meaning of the value member and its type. This meaning is two-fold:

1. The string representation of the value if its native type is not a string.
2. The UI representation used to display the value.

The value SHOULD be a member of FieldType and the importing provider SHOULD ignore any unknown values and default to string.

value

This member contains the fieldType defined by the user.

label

This member contains a user facing value describing the value stored. This value MAY be user defined.

3.5.5. FieldType Enumeration§

```
FieldType =
    "string" /
    "concealed-string" /
    "email" /
    "number" /
    "boolean" /
    "date"
```

string
A UTF-8 encoded string value which is unconcealed and does not have a specified format.

concealed-string
A UTF-8 encoded string value which should be considered secret and not displayed unless the user explicitly requests it.

email
A UTF-8 encoded string value which follows the format specified in [[!RFC5322#section-3.4]]. This field SHOULD be unconcealed.

number
A stringified numeric value which is unconcealed.

boolean
A boolean value which is unconcealed. It MUST be of the values`"true"` or `"false"`.

date
A string value representing a calendar date which follows the format specified in[RFC3339].

3.5.6. Fido2Extensions dictionary§

```
Fido2Extensions = {
    ? hmacSecret: Fido2HmacSecret,
    ? credBlob: b64url,
    ? largeBlob: Fido2LargeBlob,
    ? payments: bool,
    ? supplementalKeys: Fido2SupplementalKeys,
}
```

3.5.7. Fido2HmacSecret§

```
Fido2HmacSecret = {
    algorithm: tstr,
    secret: b64url,
}
```

3.5.8. Fido2LargeBlob§

```
Fido2LargeBlob = {
    size: uint,
    alg: tstr,
    data: b64url,
}
```

3.5.9. Fido2SupplementalKeys§

```
Fido2SupplementalKeys = {
    ? device: bool,
    ? provider: bool,
}
```

3.6. Defined **Extension**§

```
Extension = $Extension .within {
    name: tstr
    ; Should there be an included schema? or use a URI to define the schema?
}
```

name

The name of the extension which will define the contents associated. If the extension is defined in this document then the value will directly use that name. If this is a custom extension defined by the exporter, then the value MUST take the following format: `EXPORTER_RP_ID/EXTENSION_NAME`. As an example `1password.com/VaultType`.

3.6.1. Sharing an Entity (Sharing)§

```
$Extension /= Shared
Shared = {
    name: "shared",
    accessors: [ * SharingAccessor ],
}
```

3.6.1.1. ***SharingAccessor***§

```
SharingAccessor = {
    type: SharingAccessorType / tstr,
    accountId: b64url,
    name: tstr,
    permissions: [ * SharingAccessorPermission / tstr ],
}
```

type

This member specifies the type of access that the user by the accountId has to this entity. The value SHOULD be a member of SharingAccessorType but importers MUST ignore any SharingAccessor entries that are unknown values for this member.

accountId

This member points to an Account's id that has been given access to this collection by the current Account.

name

This member contains the userName if type is of value user. If type is of value group this member then contains the group's name.

permissions

This member lists the permissions that this accountId has to the associated Collection. The values SHOULD be members of SharingAccessorPermission but importers MUST ignore unknown values, ignoring any unknown values in permissions. The importer MUST ignore any SharingAccessors that have an empty permissions list, whether it's been exported as empty, or the result of ignoring all unknown values.

3.6.1.2. ***SharingAccessorType*** *Enumeration*§

```
SharingAccessorType =
    "user" /
    "group"
```

user
Indicates the respective SharingAccessor is describing a user’s permissions on the Collection.

group
Indicates the respective SharingAccessor is describing a group of users' permissions on theCollection.

3.6.1.3. ***SharingAccessorPermission*** *Enumeration*§

```
SharingAccessorPermission =
    "read" /
    "update" /
    "create" /
    "delete" /
    "share" /
    "manage"
```

read
Indicates that the respective SharingAccessor has read permissions on all Items in the associated Collection.

update
Indicates that the respective SharingAccessor has update permissions on all Items in the associated Collection.

create
Indicates that the respective SharingAccessor has the permission to create newItems in the associated Collection.

delete
Indicates that the respective SharingAccessor has the permission to delete any Item in the associated Collection

share
Indicates that the respective SharingAccessor can share any Item from the associated Collection with users or groups if they so choose.

manage
Indicates that the respective SharingAccessor can manage this Collection, meaning they can edit the collection’s attributes, share it with others, etc.

4. Usage Guidelines§

[Offer guidelines for using the CXF format to import and export credentials securely.]

4.1. Importing Credentials§

[Explain the steps and considerations for importing credentials using the CXF format.]

4.2. Exporting Credentials§

[Provide instructions for exporting credentials to the CXF format.]

5. Examples§

[Present practical examples of importing and exporting credentials using the CXF format.]

5.1. Importing a Credential Set§

[Walk through the process of importing a set of credentials using CXF.]

5.2. Exporting a Credential Set§

[Provide an example of exporting a credential set to the CXF format.]

6. IANA Considerations§

[Outline considerations related to IANA registrations, including the CXF media type.]

6.1. CXF Media Type§

[Specify the media type for CXF and its registration details.]

7. Security Considerations§

[Provide an in-depth analysis of the security aspects of the CXF format and its use.]

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§

- human-palatable
- id
- name
- private key
- public key
- rawId
- registration
- relying party
- relying party identifier
- rp id
- user account
- user handle
- webauthn

References§

Normative References§

[CXP]
N. Steele. *Credential Exchange Format*. June 20, 2024. FIDO Alliance Working Draft. URL: https://drafts.fidoalliance.org/fido-2/stable-links-to-latest/cxp.html

[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

[RFC3339]
G. Klyne; C. Newman. *Date and Time on the Internet: Timestamps*. July 2002. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc3339

[RFC5958]
S. Turner. *Asymmetric Key Packages*. August 2010. Proposed Standard. URL:https://www.rfc-editor.org/rfc/rfc5958

[W3C-PROCESS]
Elika J. Etemad (fantasai); Florian Rivoal. *W3C Process Document*. 2 November 2021. URL: https://www.w3.org/Consortium/Process/

[WebAuthn]
Dirk Balfanz (Google); et al. *Web Authentication: An API for accessing Public Key Credentials Level 2*. 8 April 2021. TR. URL: https://www.w3.org/TR/webauthn-2/