[SMART Health Cards Framework](https://spec.smarthealth.cards/" \l "certificates)

Protocol

Overview[¶](https://spec.smarthealth.cards/#overview)

**Looking for a non-technical overview?**[**¶**](https://spec.smarthealth.cards/#looking-for-a-non-technical-overview)

See the [SMART Health Cards public landing page](https://smarthealth.cards/). Otherwise, read on for the technical specifications.

**Status**[**¶**](https://spec.smarthealth.cards/#status)

Stable first release authored with input from technology, lab, pharmacy, Electronic Health Record, and Immunization Information System vendors. The current version of the framework is 1.2.1; see the revision history in the [change log](https://spec.smarthealth.cards/changelog/).

**Contributing**[**¶**](https://spec.smarthealth.cards/#contributing)

This specification is copyright by *Computational Health Informatics Program, Boston Children's Hospital, Boston, MA* and licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/).

We welcome discusion on the [SMART Health Cards channel](https://chat.fhir.org/#narrow/stream/284830-smart.2Fhealth-cards) of the FHIR community chat. You can also propose changes via GitHub [Issues](https://github.com/smart-on-fhir/health-cards/issues) or create a [Pull Request](https://github.com/smart-on-fhir/health-cards/pulls).

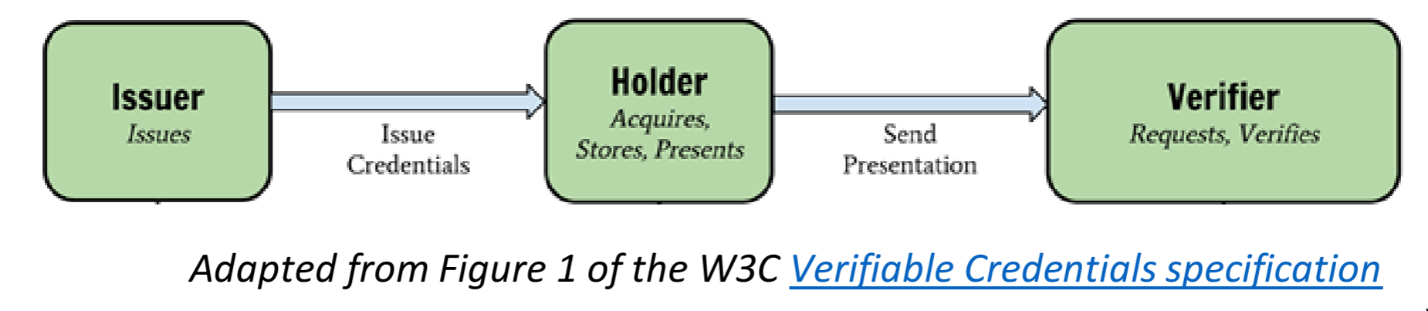
Security issues can be disclosed privately by emailing security@smarthealth.cards to allow for a responsible disclosure to affected parties.

Introduction[¶](https://spec.smarthealth.cards/#introduction)

This implementation guide provides a framework for "Health Cards", with a short term goal to enable a consumer to receive COVID-19 Vaccination or Lab results and **present these results to another party in a verifiable manner**. Key use cases include conveying point-in-time infection status for return-to-workplace and travel. This approach should also support documentation of immunization status and other health details.

Because we must ensure end-user privacy and because Health Cards must work across organizational and jurisdictional boundaries, we are building on international open standards and decentralized infrastructure.

Conceptual Model[¶](https://spec.smarthealth.cards/#conceptual-model)



* **Issuer** (e.g., a lab, pharmacy, healthcare provider, EHR, public health department, or immunization information system) generates verifiable credentials
* **Holder** stores credentials and presents them at will
* **Verifier** receives credentials from holder and ensures they are properly signed

Design Goals[¶](https://spec.smarthealth.cards/#design-goals)

* Support **end-to-end workflow** where users receive and present relevant healthcare data
* Enable workflow with **open standards**
* Support strong **cryptographic signatures**
* Enable **privacy preserving** data presentations for specific use cases

Start Small -- Think Big[¶](https://spec.smarthealth.cards/#start-small-think-big)

We enable Health Cards by defining building blocks that can be used across healthcare. The core building block allows us to aggregate data into meaningful sets, signed by an issuer, and stored/presented by a consumer as needed. The broader set of use cases might eventually include:

* Managing an immunization record that can be shared with schools or employers, or for travel
* Sharing verifiable health history data with clinical research studies
* Sharing voluntary data with public health agencies
* Sharing questionnaire responses with healthcare providers

Despite this broad scope, our *short-term definition of success* requires that we:

* Represent "Health Cards" in a "Health Wallet", focusing on COVID-19 status
* Ensure that each role (issuer, holder, app) can be implemented by any organization following open standards, provided they sign on to the relevant trust framework

User Experience and Data Flow[¶](https://spec.smarthealth.cards/#user-experience-and-data-flow)

* **User Receives** a Health Card from an Issuer. The Health Card is a signed data artifact that the user can obtain through any of these methods:
  + issuer offers a Health Card on paper or PDF, including a QR code (required method)
  + issuer offers a Health Card for download as a .smart-health-card file (required method)
  + issuer hosts a Health Card for [FHIR API access](https://spec.smarthealth.cards/#healthwalletissuevc-operation) via a compatible Health Wallet application. This workflow includes a SMART on FHIR authorization step with an Issuer, where the user grants read access to any resources that will be present in Health Cards (e.g., Patient, Immunization, Observation, DiagnosticReport)
* **User Saves** a Health Card, whether on paper or digitally.
* **User Presents** a Health Card to a Verifier. Presentation includes explicit user opt-in and approval, and may involve displaying a QR code, sharing a file, or using an on-device SDK (e.g., for verifier-to-holder app-to-app communications)

Trust[¶](https://spec.smarthealth.cards/#trust)

Anyone can *issue* Health Cards, and every verifier can make its own decision about which issuers to *trust*. A "trust framework" can help verifiers to externalize these decisions and drive toward more consistent practices. The SMART Health Cards IG is designed to operate independent of any trust framework, while allowing trust frameworks to be layered on top. We anticipate such frameworks will emerge to meet different jurisdictional and use case driven requirements. In all cases, verifiers can discover public keys associated with an issuer via /.well-known/jwks.json URLs.

Privacy[¶](https://spec.smarthealth.cards/#privacy)

**Data Minimization**[**¶**](https://spec.smarthealth.cards/#data-minimization)

It is an explicit design goal to let the holder **only disclose a minimum amount of information** to a verifier. The information *required* to be disclosed is use-case dependent, and -- particularly in a healthcare setting -- it can be difficult for lay people to judge which data elements are necessary to be shared.

The granularity of information disclosure will be at the level of an entire credential (i.e., a user can select "which cards" to share from a Health Wallet, and each card is shared wholesale). The credentials are designed to only include the minimum information necessary for a given use case.

**Granular Sharing**[**¶**](https://spec.smarthealth.cards/#granular-sharing)

Data holders should have full control over the data they choose to share for a particular use-case. Since Health Cards are signed by the issuer and cannot be altered later, it is important to ensure that Health Cards are created with granular sharing in mind. Therefore, issuers SHOULD only combine distinct data elements into a Health Card when a Health Card FHIR profile requires it.

Additionally, Health Card FHIR Profiles SHOULD only include data that need to be conveyed together. (e.g., immunizations for different diseases should be kept separate. Immunizations and lab results should be kept separate.)

**Future Considerations**[**¶**](https://spec.smarthealth.cards/#future-considerations)

If we identify *optional* data elements for a given use case, we might incorporate them into credentials by including a cryptographic hash of their values instead of embedding values directly. Longer term we can provide more granular options using techniques like zero-knowledge proofs, or by allowing a trusted intermediary to summarize results in a just-in-time fashion.

Data Model[¶](https://spec.smarthealth.cards/#data-model)

This framework defines a general approach to **representing demographic and clinical data in FHIR**, outlined in [Modeling Verifiable Credentials in FHIR](https://spec.smarthealth.cards/credential-modeling/). Specific use cases for Health Cards will define specific data profiles.

* **COVID-19 Vaccination Credentials**: See [SMART Health Cards: Vaccination IG](http://vci.org/ig/vaccination-and-testing)

Protocol Details[¶](https://spec.smarthealth.cards/#protocol-details)

Generating and resolving cryptographic keys[¶](https://spec.smarthealth.cards/#generating-and-resolving-cryptographic-keys)

The following key types are used in the Health Cards Framework:

* Elliptic Curve keys using the P-256 curve

**Signing *Health Cards***[**¶**](https://spec.smarthealth.cards/#signing-health-cards)

* Issuers sign Health Card VCs (Verifiable Credentials) with a signing key (private key)
* Issuer publish the corresponding public key (public key) at /.well-known/jwks.json
* Wallets and Verifiers use the public key to verify Issuer signatures on Health Cards

**Determining keys associated with an issuer**[**¶**](https://spec.smarthealth.cards/#determining-keys-associated-with-an-issuer)

Each public key used to verify signatures is represented as a JSON Web Key (see [RFC 7517](https://tools.ietf.org/html/rfc7517)):

* SHALL have "kty": "EC", "use": "sig", and "alg": "ES256"
* SHALL have "kid" equal to the base64url-encoded SHA-256 JWK Thumbprint of the key (see [RFC7638](https://tools.ietf.org/html/rfc7638))
* SHALL have "crv": "P-256, and "x", "y" equal to the base64url-encoded values for the public Elliptic Curve point coordinates (see [RFC7518](https://tools.ietf.org/html/rfc7518#section-6.2))
* SHALL NOT have the Elliptic Curve private key parameter "d"
* If the issuer has an X.509 certificate for the public key, SHALL have "x5c" equal to an array of one or more base64-encoded (not base64url-encoded) DER representations of the public certificate or certificate chain (see [RFC7517](https://tools.ietf.org/html/rfc7517#section-4.7)). The public key listed in the first certificate in the "x5c" array SHALL match the public key specified by the "crv", "x", and "y" parameters of the same JWK entry. If the issuer has more than one certificate for the same public key (e.g. participation in more than one trust community), then a separate JWK entry is used for each certificate with all JWK parameter values identical except "x5c".

Issuers SHALL publish their public keys as JSON Web Key Sets (see [RFC7517](https://tools.ietf.org/html/rfc7517#section-5)), available at <<iss value from JWS>> + /.well-known/jwks.json, with [Cross-Origin Resource Sharing (CORS)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin) enabled, using TLS version 1.2 following the IETF [BCP 195](https://www.rfc-editor.org/info/bcp195) recommendations or TLS version 1.3 (with any configuration).

The URL at <<iss value from JWS>> SHALL use the https scheme and SHALL NOT include a trailing /. For example, https://smarthealth.cards/examples/issuer is a valid iss value (https://smarthealth.cards/examples/issuer/ is **not**).

**Signing keys** in the .keys[] array can be identified by kid following the requirements above (i.e., by filtering on kty, use, and alg).

For example, the following is a fragment of a jwks.json file with one signing key:

{

"keys":[

{

"kty": "EC",

"kid": "\_IY9W2kRRFUigDfSB9r8jHgMRrT0w4p5KN93nGThdH8",

"use": "sig",

"alg": "ES256",

"crv": "P-256",

"x": "7xbC\_9ZmFwKqOHpwX6-LnlhIh5SMIuNwl0PW1yVI\_sk",

"y": "7k2fdIRNDHdf93vL76wxdXEPtj\_GiMTTyecm7EUUMQo",

}

]

}

**Certificates**[**¶**](https://spec.smarthealth.cards/#certificates)

X.509 certificates can be used by issuers to indicate the issuer's participation in a PKI-based trust framework.

If the Verifier supports PKI-based trust frameworks and the Health Card issuer includes the "x5c" parameter in matching JWK entries from the .keys[] array, the Verifier establishes that the issuer is trusted as follows:

1. Verifier validates the leaf certificate's binding to the Health Card issuer by:
   * matching the <<iss value from JWS>> to the value of a uniformResourceIdentifier entry in the certificate's Subject Alternative Name extension (see [RFC5280](https://tools.ietf.org/html/rfc5280#section-4.2.1.6)), and
   * verifying the signature in the Health Card using the public key in the certificate.
2. Verifier constructs a valid certificate path of unexpired and unrevoked certificates to one of its trusted anchors (see [RFC5280](https://tools.ietf.org/html/rfc5280#section-6)).

**Key Management**[**¶**](https://spec.smarthealth.cards/#key-management)

Issuers SHOULD generate new signing keys at least annually.

When an issuer generates a new key to sign Health Cards, the public key SHALL be added to the issuer's JWK set in its jwks.json file. Retired private keys that are no longer used to sign Health Cards SHALL be destroyed. Older public key entries that are needed to validate previously signed Health Cards SHALL remain in the JWK set for as long as the corresponding Health Cards are clinically relevant. However, if a private signing key is compromised, then the issuer SHALL immediately remove the corresponding public key from the JWK set in its jwks.json file and request revocation of all X.509 certificates bound to that public key; verifiers will from then on reject all Health Cards signed using that key.

**Revocation**[**¶**](https://spec.smarthealth.cards/#revocation)

Individual Health Cards MAY be revoked using a revocation identifier property rid encoded in the vc claim of the JWT. This should be a short identifier, meaningless to the verifiers; the only constraint is that the identifier SHALL use the base64url alphabet (but doesn’t need to be base64url encoded) and be no longer than 24 characters. Issuers MAY use application-specific user identifiers for this purpose, but since these could be publicly listed in revocation lists, issuers SHOULD use a one-way transformation of the data combined with enough entropy to prevent reversal. It is RECOMMENDED to use the base64url encoding of the first 64 bits of the output of HMAC-SHA-256 (as specified in [RFC 4868](https://tools.ietf.org/html/rfc4868)) on the user identifier using a 256-bit random secret key concatenated with the <<kid>>; i.e., `rid = base64url(hmac-sha-256(secret\_key || <>, user\_id)[1..64]).

To enable per-card revocation, the issuer creates, for each of its keys, a JSON Card Revocation List (CRL) file with the following content:

{

"kid": "<<kid>>",

"method": "rid",

"ctr": "<<ctr>>",

"rids": [...]

}

where - "<<kid>>" is the ID of the corresponding issuer key, - "rid" identifies the revocation method specified in this framework; legacy cards can use different methods specified in external revocation profiles, - "<<ctr>>" is a counter indicating how many times this file has been updated; initial value is 1, - rids is an array of revoked cards' identifiers rid values. These values are represented as strings from the base64url alphabet, plus an optional timestamp suffix consisting of . followed by a numerical timestamp (e.g., .1636977600)

To revoke a Health Card issued under the key "<<kid>>", an issuer adds its revocation identifier to the rids array of the corresponding <<kid>>'s revocation file. Since an issuer might want to invalidate a series of Health Cards associated with the user up to a certain time, the rid might be followed by a separator . a timestamp (encoded as the number of seconds from 1970-01-01T00:00:00Z UTC, as specified by RFC 7519). After updating the rids array (with one or more items), the <<ctr>> is incremented.

As an example, the rids array ["AQPCj4wwk6Mt", "lHKzqFUMjhs.1636977600"] marks as revoked any Health Cards with rid equal to AQPCj4wwk6Mt and Health Cards with rid equal to lHKzqFUMjhs issued before November 15, 2021 12:00:00 PM GMT.

The per-key revocation file is made available at https://"<<Issuer URL>>"/.well-known/crl/"<<kid>>".json, where - "<<Issuer URL>>" is the issuer URL listed in the Health Card, - "<<kid>>" is the key ID with which the Health Card was signed.

Issuers supporting this revocation method SHALL include in their published JWK set, for each key, a crlVersion field encoding the update counter "<>" for the corresponding revocation file.

If the crlVersion is present in the Issuer's JWK for key <<kid>>, Verifiers SHALL - Download the https://"<<Issuer URL>>"/.well-known/crl/"<<kid>>".json file or use a cached version if the counter value has not changed since the last retrieval, - Reject the Health Card if the calculated rid is contained in the CRL's rids array and (if a timestamp suffix is present) the Health Card’s nbf is value is before the timestamp.

Revocation of Health Cards without a rid field (including all pre-v1.2.0 ones) can be done using external mechanisms to calculate a dynamic rid value based on the JWS’s content.

If individual revocation of SMART Health Cards is not possible, then an issuer SHOULD revoke its issuing key, and allow users to obtain new Health Cards; limiting the validity period of a key helps to mitigate the adverse effects of this situation. See the [revocation FAQ](https://github.com/smart-on-fhir/health-cards/blob/main/FAQ/revocation.md) for more details.

Issuer Generates Results[¶](https://spec.smarthealth.cards/#issuer-generates-results)

When the issuer is ready to generate a Health Card, the issuer creates a FHIR payload and packs it into a corresponding Health Card VC (or Health Card Set).

HolderIssuerEarlier...Generate Issuer's keysIf Health Card data for holder already exist: re-generate VCsData CreatedGenerate FHIR RepresentationGenerate VC RepresentationGenerate JWS Payload and signLater...Holder receives Health CardHolderIssuer

**Health Cards are encoded as Compact Serialization JSON Web Signatures (JWS)**[**¶**](https://spec.smarthealth.cards/#health-cards-are-encoded-as-compact-serialization-json-web-signatures-jws)

The VC structure (scaffold) is shown in the following example. The Health Cards framework serializes VCs using the compact JWS serialization, where the payload is a compressed set of JWT claims (see [Appendix 3 of RFC7515](https://tools.ietf.org/html/rfc7515#appendix-A.3) for an example using ECDSA P-256 SHA-256, as required by this specification). Specific encoding choices ensure compatibility with standard JWT claims, as described at <https://www.w3.org/TR/vc-data-model/#jwt-encoding>.

The type, and credentialSubject properties are added to the vc claim of the JWT. The type values are defined in [Credential Types](https://smarthealth.cards/vocabulary/); the https://smarthealth.cards#health-card SHALL be present; other types SHOULD be included when they apply. Verifiers and other entities processing SMART Health Cards SHALL ignore any additional type elements they do not understand. The issuer property is represented by the registered JWT iss claim and the issuanceDate property is represented by the registered JWT nbf ("not before") claim (encoded as the number of seconds from 1970-01-01T00:00:00Z UTC, as specified by [RFC 7519](https://tools.ietf.org/html/rfc7519)). Hence, the overall JWS payload matches the following structure (before it is [minified and compressed](https://spec.smarthealth.cards/#health-cards-are-small)):

{

"iss": "<<Issuer URL>>",

"nbf": 1591037940,

"vc": {

"type": [

"https://smarthealth.cards#health-card",

"<<Additional Types>>",

],

"credentialSubject": {

"fhirVersion": "<<FHIR Version, e.g. '4.0.1'>>",

"fhirBundle":{

"resourceType": "Bundle",

"type": "collection",

"entry": ["<<FHIR Resource>>", "<<FHIR Resource>>", "..."]

}

}

}

}

**Health Cards are Small**[**¶**](https://spec.smarthealth.cards/#health-cards-are-small)

To ensure that all Health Cards can be represented in QR codes, issuers SHALL ensure that the following constraints apply at the time of issuance:

* JWS Header
  + header includes alg: "ES256"
  + header includes zip: "DEF"
  + header includes kid equal to the base64url-encoded SHA-256 JWK Thumbprint of the key (see [RFC7638](https://tools.ietf.org/html/rfc7638))
* JWS Payload
  + payload is minified (i.e., all optional whitespace is stripped)
  + payload is compressed with the DEFLATE (see [RFC1951](https://www.ietf.org/rfc/rfc1951.txt)) algorithm before being signed (note, this should be "raw" DEFLATE compression, omitting any zlib or gz headers)
  + payload .vc.credentialSubject.fhirBundle is created:
    - without Resource.id elements
    - without Resource.meta elements (or if present, .meta.security is included and no other fields are included)
    - without DomainResource.text elements
    - without CodeableConcept.text elements
    - without Coding.display elements
    - with Bundle.entry.fullUrl populated with short resource-scheme URIs (e.g., {"fullUrl": "resource:0"})
    - with Reference.reference populated with short resource-scheme URIs (e.g., {"patient": {"reference": "resource:0"}})

For details about how to embed Health Cards in a QR code, [see below](https://spec.smarthealth.cards/#every-health-card-can-be-embedded-in-a-qr-code).

User Retrieves Health Cards[¶](https://spec.smarthealth.cards/#user-retrieves-health-cards)

In this step, the user learns that a new Health Card is available (e.g., by receiving a text message or email notification, or by an in-wallet notification for FHIR-enabled issuers.)

**via File Download**[**¶**](https://spec.smarthealth.cards/#via-file-download)

To facilitate this workflow, the issuer can include a link to help the user download the credentials directly, e.g., from a login-protected page in the Issuer's patient portal. The file SHALL be served with a .smart-health-card file extension and SHALL be provided with a MIME type of application/smart-health-card (e.g., web servers SHALL include Content-Type: application/smart-health-card as an HTTP Response containing a Health Card), so the Health Wallet app can be configured to recognize this extension and/or MIME type. Contents should be a JSON object containing an array of Verifiable Credential JWS strings:

{

"verifiableCredential": [

"<<Verifiable Credential as JWS>>",

"<<Verifiable Credential as JWS>>"

]

}

**via QR (Print or Scan)**[**¶**](https://spec.smarthealth.cards/#via-qr-print-or-scan)

Alternatively, issuers can make any individual JWS inside a Health Card available **embedded in a QR code** (for instance, printed on a paper-based vaccination record or after-visit summary document). See [details](https://spec.smarthealth.cards/#every-health-card-can-be-embedded-in-a-qr-code).

Finally, the Health Wallet asks the user if they want to save any/all of the supplied credentials.

**via FHIR $health-cards-issue Operation**[**¶**](https://spec.smarthealth.cards/#via-fhir-health-cards-issue-operation)

For a more seamless user experience when FHIR API connections are already in place, results may also be conveyed through a FHIR API $health-cards-issue operation defined [here](https://spec.smarthealth.cards/artifacts/operation-patient-i-health-cards-issue.json). For issuers that support SMART on FHIR access, the Health Wallet MAY request authorization with SMART on FHIR scopes (e.g., launch/patient patient/Immunization.read for an Immunization use case). This allows the Health Wallet to automatically request issuance of VCs, including requests for periodic updates.

**DISCOVERY OF FHIR SUPPORT**

A SMART on FHIR Server capable of issuing VCs according to this specification SHALL advertise its support by adding the health-cards capability to its /.well-known/smart-configuration JSON file. For example:

{

"authorization\_endpoint": "https://ehr.example.com/auth/authorize",

"token\_endpoint": "https://ehr.example.com/auth/token",

"token\_endpoint\_auth\_methods\_supported": ["client\_secret\_basic"],

"scopes\_supported": ["launch", "launch/patient", "patient/\*.\*", "offline\_access"],

"response\_types\_supported": ["code", "code id\_token", "id\_token", "refresh\_token"],

"capabilities": ["health-cards", "launch-standalone", "context-standalone-patient", "client-confidential-symmetric"]

}

**$HEALTH-CARDS-ISSUE OPERATION**

A Health Wallet can POST /Patient/:id/$health-cards-issue to a FHIR-enabled issuer to request the generation of a specific type of Health Card. The body of the POST looks like:

{

"resourceType": "Parameters",

"parameter": [{

"name": "credentialType",

"valueUri": "https://smarthealth.cards#covid19"

}]

}

The credentialType parameter is required. Multiple credentialType values in one request SHALL be interpreted as a request for the intersection of the requested types (logical AND). For example, a request containing credentialType values https://smarthealth.cards#covid19 and https://smarthealth.cards#immunization is a request for only those cards that are both COVID-19 cards and immunization cards (i.e., only those COVID-19 cards that are about immunizations).

The following parameters are optional; clients MAY include them in a request, and servers MAY ignore them if present.

* **includeIdentityClaim**. By default, the issuer will decide which identity claims to include, based on profile-driven guidance. If the Health Wallet wants to fine-tune identity claims in the generated credentials, it can provide an explicit list of one or more includeIdentityClaims, which will limit the claims included in the VC. For example, to request that only name be included:

{

"resourceType": "Parameters",

"parameter": [{

"name": "credentialType",

"valueUri": "https://smarthealth.cards#covid19"

}, {

"name": "includeIdentityClaim",

"valueString": "Patient.name"

}]

}

* **\_since**. By default, the issuer will return Health Cards of any age. If the Health Wallet wants to request only cards pertaining to data since a specific point in time, it can provide a \_since parameter with a valueDateTime (which is an ISO8601 string at the level of a year, month, day, or specific time of day using the extended time format; see [FHIR dateTime datatype](http://hl7.org/fhir/datatypes.html#dateTime) for details). For example, to request only COVID-19 data since March 2021:

{

"resourceType": "Parameters",

"parameter": [{

"name": "credentialType",

"valueUri": "https://smarthealth.cards#covid19"

}, {

"name": "\_since",

"valueDateTime": "2021-03"

}]

}

The **response** is a Parameters resource that includes one more more verifiableCredential values like:

{

"resourceType": "Parameters",

"parameter":[{

"name": "verifiableCredential",

"valueString": "<<Health Card as JWS>>"

}]

}

If no results are available, a Parameters resource without any parameter is returned:

{

"resourceType": "Parameters"

}

In the response, an optional repeating resourceLink parameter can capture the link between any number of hosted FHIR resources and their derived representations within the verifiable credential's .credentialSubject.fhirBundle, allowing the health wallet to explicitly understand these correspondences between bundledResource and hostedResource, without baking details about the hosted endpoint into the signed credential. The optional vcIndex value on a resourceLink can be used when a response contains more than one VC, to indicate which VC this resource link applies to. The vcIndex is a zero-based index of a verifiableCredential entry within the top-level parameter array.

{

"resourceType": "Parameters",

"parameter": [{

"name": "verifiableCredential",

"valueString": "<<Health Card as JWS>>"

}, {

"name": "resourceLink",

"part": [{

"name": "vcIndex",

"valueInteger": 0

}, {

"name": "bundledResource",

"valueUri": "resource:2"

}, {

"name": "hostedResource",

"valueUri": "https://fhir.example.org/Immunization/123"

}]

}]

}

Presenting Health Cards to a Verifier[¶](https://spec.smarthealth.cards/#presenting-health-cards-to-a-verifier)

In this step, the verifier asks the user to share a COVID-19 result. A Health Card containing the result can be conveyed by presenting a QR code; by uploading a file; or by leveraging device-specific APIs. Over time, we will endeavor to standardize presentation workflows including device-specific patterns and web-based exchange.

Every Health Card can be embedded in a QR code[¶](https://spec.smarthealth.cards/#every-health-card-can-be-embedded-in-a-qr-code)

Each JWS string that appears in the .verifiableCredential[] of a .smart-health-card file can be embedded in one or more QR codes. We aim to ensure that printed (or electronically displayed) codes are usable at physical dimensions of 40mmx40mm. This constraint allows us to use QR codes up to Version 22, at 105x105 modules. When embedding a JWS string in QR codes, the JWS string SHALL be encoded as Numerical Mode QR codes consisting of the digits 0-9 (see ["Encoding Chunks as QR codes"](https://spec.smarthealth.cards/#encoding-chunks-as-qr-codes)).

Ensuring Health Cards can be presented as QR codes:

* Allows basic storage and sharing of Health Cards for users without a smartphone
* Allows smartphone-enabled users to print a usable backup
* Allows full Health Card contents to be shared with a verifier

The following limitations apply when presenting Health Card as QR codes, rather than engaging in device-based workflows:

* Does not capture a digital record of a request for presentation
* Verifier cannot include requirements in-band
* Verifier cannot include purposes of use in-band
* Does not capture a digital record of the presentation

Creating a QR code (or a set of QR codes) from a Health Card JWS[¶](https://spec.smarthealth.cards/#creating-a-qr-code-or-a-set-of-qr-codes-from-a-health-card-jws)

**Chunking**[**¶**](https://spec.smarthealth.cards/#chunking)

Commonly, Health Cards will fit in a single V22 QR code. Any JWS longer than 1195 characters SHALL be split into "chunks" of length 1191 or smaller; each chunk SHALL be encoded as a separate QR code of V22 or lower, to ensure ease of scanning. Each chunk SHALL be numerically encoded and prefixed with an ordinal as well as the total number of chunks required to re-assemble the JWS, as described below. The [QR code FAQ page](https://github.com/smart-on-fhir/health-cards/blob/main/FAQ/qr.md) details max JWS length restrictions at various error correction levels.

To ensure the best user experience when producing and consuming multiple QR codes:

* Producers of QR codes SHOULD balance the sizes of chunks. For example, if a JWS is 1200 characters long, producers should create two ~600 character chunks rather than a 1191 character chunk and a 9 character chunk.
* Consumers of QR codes SHOULD allow for scanning the multiple QR codes in any order. Once the full set is scanned, the JWS can be assembled and validated.

**Encoding Chunks as QR codes**[**¶**](https://spec.smarthealth.cards/#encoding-chunks-as-qr-codes)

When printing or displaying a Health Card using QR codes, let "N" be the total number of chunks required, and let "C" be a variable indicating the index of the current chunk. Each chunk of the JWS string value SHALL be represented as a QR with two data segments:

1. A segment encoded with bytes mode consisting of
   * the fixed string shc:/ (registered as an [IANA scheme](https://www.iana.org/assignments/uri-schemes/prov/shc))
   * plus (only if more than one chunk is required)
     + decimal representation of "C" (e.g., 1 for the first chunk, 2 for the second chunk, and so on)
     + plus the fixed string /
     + plus decimal representation of "N" (e.g., 2 if there are two chunks in total, 3 if there three chunks in total, and so on)
     + plus the fixed string /
2. A segment encoded with numeric mode consisting of the characters 0-9. Each character "c" of the JWS is converted into a sequence of two digits as by taking Ord(c)-45 and treating the result as a two-digit base ten number. For example, 'X' is encoded as 43, since Ord('X') is 88, and 88-45 is 43. (The constant "45" appears here because it is the ordinal value of -, the lowest-valued character that can appear in a compact JWS. Subtracting 45 from the ordinal values of valid JWS characters produces a range between 00 and 99, ensuring that each character of the JWS can be represented in exactly two base-10 numeric digits.)

(The reason for representing Health Cards using Numeric Mode QRs instead of Binary Mode (Latin-1) QRs is information density: with Numeric Mode, 20% more data can fit in a given QR, vs Binary Mode. This is because the JWS character set conveys only log\_2(65) bits per character (~6 bits); binary encoding requires log\_2(256) bits per character (8 bits), which means ~2 wasted bits per character.)

For example:

* a single chunk might produce a QR code like shc:/56762909524320603460292437404460<snipped for brevity>
* in a longer JWS, the second chunk in a set of three might produce a QR code like shc:/2/3/56762909524320603460292437404460<snipped for brevity>

When reading a QR code, scanning software can recognize a SMART Health Card from the shc:/ prefix. Stripping this prefix and the following <ordinal>/<count>/ and decoding the remaining pairs of numerals yields a JWS.

Expiration of Health Cards[¶](https://spec.smarthealth.cards/#expiration-of-health-cards)

SMART Health Cards contain factual information that is assured to be correct at the point of issuance and does not change with the passage of time. Therefore, **Health Cards generally do not expire** and an expiration date is not used. There are, however, situations where the ability to set an expiration date is beneficial.

One use case for issuing SMART Health Cards with an expiration date is a government entity issuing a vaccination card to foreign visitors for their use while in the destination country. This visitor's vaccination card is issued based on original documents presented by the visitor. Even with robust verification protocols, the government entity may not want to vouch for the validity of the visitor pass for an unlimited period of time. Importantly, the original document may be invalidated at some point in the future, e.g. by its signing keys being revoked. It may be impractical for the government entity issuing the visitor pass to track and reactively revoke the visitor pass. This risk can be mitigated by setting an expiration date on the visitor pass at the time of issuance. The expiration date could, for example, correspond to the visitor's allowed duration of stay in the foreign country.

To address use cases such as the preceding one, an optional SMART Health Card expiration date can be represented by the registered JWT exp claim (encoded as the number of seconds from 1970-01-01T00:00:00Z UTC, as specified by [RFC 7519](https://tools.ietf.org/html/rfc7519)). Verifiers SHALL check the expiration, if present, and reject SMART Health Cards with an exp value that is before the current verification date-time.

FAQ[¶](https://spec.smarthealth.cards/#faq)

Technical security questions are covered in the [security FAQ page](https://github.com/smart-on-fhir/health-cards/blob/main/FAQ/security.md).

Can a SMART Health Card be used as a form of identification?[¶](https://spec.smarthealth.cards/#can-a-smart-health-card-be-used-as-a-form-of-identification)

No. SMART Health Cards are designed for use *alongside* existing forms of identification (e.g., a driver's license in person, or an online ID verification service). A SMART Health Card is a non-forgeable digital artifact analogous to a paper record on official letterhead. Concretely, the problem SMART Health Cards solve is one of provenance: a digitally signed SMART Health Card is a credential that guarantees that a specific issuer generated the record. The duty of verifying that the person presenting a Health Card *is* the subject of the data within the Health Card (or is authorized to act on behalf of this data subject) falls to the person or system receiving and validating a Health Card.

Which clinical data should be considered in decision-making?[¶](https://spec.smarthealth.cards/#which-clinical-data-should-be-considered-in-decision-making)

* The data in Health Cards should focus on communicating "immutable clinical facts".
* Each use case will define specific data profiles.
  + For COVID-19 Vaccination Credentials, the [SMART Health Cards: Vaccination IG](http://vci.org/ig/vaccination-and-testing) defines requirements.
* When Health Cards are used in decision-making, the verifier is responsible for deciding what rules to apply. For example:
  + decision-making rules may change over time as our understanding of the clinical science improves.
  + decision-making rules may be determined or influenced by international, national and local health authorities.
  + decision-making rules may require many inputs, some of which can be supplied by Health Cards and others of which may come from elsewhere (e.g., by asking the user "are you experiencing any symptoms today?").

How can we share conclusions like a "Safe-to-fly Pass", instead of sharing clinical results?[¶](https://spec.smarthealth.cards/#how-can-we-share-conclusions-like-a-safe-to-fly-pass-instead-of-sharing-clinical-results)

Decision-making often results in a narrowly-scoped "Pass" that embodies conclusions like "Person X qualifies for international flight between Country A and Country B, according to Rule Set C". While Health Cards are designed to be long-lived and general-purpose, Passes are highly contextual. We are not attempting to standardize "Passes" in this framework, but Health Cards can provide an important verifiable input for the generation of Passes.

What testing tools are available to validate SMART Health Cards implementations?[¶](https://spec.smarthealth.cards/#what-testing-tools-are-available-to-validate-smart-health-cards-implementations)

The following tools are helpful to validate Health Card artifacts:

* The [HL7 FHIR Validator](https://confluence.hl7.org/display/FHIR/Using+the+FHIR+Validator) can be used to validate the Health Card's FHIR bundle
* The [Health Cards Dev Tools](https://github.com/smart-on-fhir/health-cards-dev-tools) can be used to validate the various Health Card artifacts.

Other resources that are helpful for learning about and implementing SMART Health Cards include:

* The [code used to generate the examples](https://github.com/smart-on-fhir/health-cards/tree/main/generate-examples) present in the spec.
* A [Jupyter Notebook walkthrough](https://github.com/dvci/health-cards-walkthrough/blob/main/SMART%20Health%20Cards.ipynb) and [demo portals](https://demo-portals.smarthealth.cards/) which demonstrate creating, validating and decoding a SMART Health Card as a QR code.

What software libraries are available to work with SMART Health Cards?[¶](https://spec.smarthealth.cards/#what-software-libraries-are-available-to-work-with-smart-health-cards)

The [Libraries for SMART Health Cards](https://github.com/smart-on-fhir/health-cards/wiki/Libraries-for-SMART-Health-Cards) wiki page includes suggestions about useful libraries.

Potential Extensions[¶](https://spec.smarthealth.cards/#potential-extensions)

**Standardized presentation workflows**[**¶**](https://spec.smarthealth.cards/#standardized-presentation-workflows)

The spec is currently focused on representing Health Cards in a standardized data payload. This allows many simple patterns for sharing, but future work can introduce standardized presentation exchange flows (e.g., OpenID Self-Issued Identity Provider, a.k.a. SIOP)

References[¶](https://spec.smarthealth.cards/#references)

* Fast Health Interoperability Resources (FHIR): <https://hl7.org/fhir/>
* DEFLATE Compression: <https://tools.ietf.org/html/rfc1951>
* JSON Web Token (JWT): <https://tools.ietf.org/html/rfc7519>
* JSON Web Key (JWK): <https://tools.ietf.org/html/rfc7517>
* JSON Web Key (JWK) Thumbprint: <https://tools.ietf.org/html/rfc7638>
* HMAC-SHA-256: <https://tools.ietf.org/html/rfc4868>

Credential Modeling

Verifiable Clinical Information in FHIR[¶](https://spec.smarthealth.cards/credential-modeling/#verifiable-clinical-information-in-fhir)

This document describes how clinical information, modeled in [FHIR](https://hl7.org/fhir), can be presented in a form based on [W3C Verifiable Credentials](https://w3c.github.io/vc-data-model/) (VC).

Content Definition[¶](https://spec.smarthealth.cards/credential-modeling/#content-definition)

Any time we want to present verifiable clinical information, we must first make some use-case-specific decisions:

1. Define a set of required and optional **FHIR content resources** (e.g., Immunization or Observation) that must be packaged and presented together
2. Decide how to bind these FHIR content resources to a person's external identity, via **FHIR identity resources** (e.g., Patient)

Once we make these decisions, we can construct a VC with a **credential subject** as follows:

* credentialSubject with these top level elements:
  + fhirVersion: a string representation of the semantic FHIR version the content is represented in (e.g. 1.0.\* for DSTU2, 4.0.\* for R4, where \* is a number, not a literal asterisk)
  + fhirBundle: a FHIR Bundle resource of type "collection" that includes all required FHIR resources (content + identity resources)

Resulting payload for the "credentialSubject":

{

"...",

"fhirVersion": "4.0.1",

"fhirBundle": {

"resourceType": "Bundle",

"type": "collection",

"entry": [

"..."

]

}

Below we focus on the Health Card use case, but the same approach to forming VCs out of FHIR can be applied to other use cases, too.

Modeling a "Health Card"[¶](https://spec.smarthealth.cards/credential-modeling/#modeling-a-health-card)

A "Health Card" is a VC that conveys results about one discrete topic -- **in this example, a COVID-19 immunization card**, encompassing details about doses given. Other cards could convey details of a RT-PCR test for COVID-19, a clinical diagnosis of COVID-19, TDAP vaccination, and so on.

According to the procedure above, we start with decisions about FHIR content resources and identity resources:

* Which **FHIR content resources** need to be conveyed in a package? For the immunization example, we'd need:
  + Immunization with details about a first dose (product, date of administration, and administering provider)
  + Immunization with details about a second dose (product, date of administration, and administering provider)
* What **FHIR identity resources** do we need to bind the FHIR content resources to an external identity system? We might eventually define use-case-specific requirements, but we want to start with a recommended set of data elements for inclusion using the FHIR Patient resource. Resources MAY include an overall "level of assurance" indicating whether these demographic elements have been verified.
  + Best practices
    - Verifiers should not store identity data conveyed via VC, and should delete data as soon as they are no longer needed for verification purposes
    - Verifiers should not expect all elements in the VC to exactly match their own records, but can still use elements conveyed in the VC.

Mapping into the W3C VC Data Model[¶](https://spec.smarthealth.cards/credential-modeling/#mapping-into-the-w3c-vc-data-model)

To create a structure matching the W3C Verifiable Credential [JSON-LD Syntax](https://www.w3.org/TR/vc-data-model/#json-ld) from a SMART Health Card JWS:

1. De-compress the JWS payload
2. Add to the .vc object:

"@context": [

"https://www.w3.org/2018/credentials/v1",

{

"@vocab": "https://smarthealth.cards#",

"fhirBundle": {

"@id": "https://smarthealth.cards#fhirBundle",

"@type": "@json"

}

}

]

1. Prepend to the .vc.type array: "VerifiableCredential"
2. Process the payload according to [JWT Decoding Rules](https://www.w3.org/TR/vc-data-model/#jwt-decoding)

**Health Card Examples**[**¶**](https://spec.smarthealth.cards/credential-modeling/#health-card-examples)

* [Example VC payloads](https://smarthealth.cards/examples/)

[PreviousProtocol](https://spec.smarthealth.cards/)

[NextCredential Types](https://spec.smarthealth.cards/vocabulary/)

Made with [Material for MkDocs](https://squidfunk.github.io/mkdocs-material/)

Credential Types

Verifiable Credential (VC) Types[¶](https://spec.smarthealth.cards/vocabulary/#verifiable-credential-vc-types)

* https://smarthealth.cards#health-card: A VC designed to convey a "Health Card" (i.e. clinical data bound to a subject's identity)

**More Granular Sub-types**[**¶**](https://spec.smarthealth.cards/vocabulary/#more-granular-sub-types)

* https://smarthealth.cards#covid19: A Health Card designed to convey COVID-19 details
* https://smarthealth.cards#immunization: A Health Card designed to convey immunization details
* https://smarthealth.cards#laboratory: A Health Card designed to convey laboratory results

Examples

Example Resources[¶](https://spec.smarthealth.cards/examples/#example-resources)

Example 0[¶](https://spec.smarthealth.cards/examples/#example-0)

* [example-00-a-fhirBundle.json](https://spec.smarthealth.cards/examples/example-00-a-fhirBundle.json)
* [example-00-b-jws-payload-expanded.json](https://spec.smarthealth.cards/examples/example-00-b-jws-payload-expanded.json)
* [example-00-c-jws-payload-minified.json](https://spec.smarthealth.cards/examples/example-00-c-jws-payload-minified.json)
* [example-00-d-jws.txt](https://spec.smarthealth.cards/examples/example-00-d-jws.txt)
* [example-00-e-file.smart-health-card](https://spec.smarthealth.cards/examples/example-00-e-file.smart-health-card)
* [example-00-f-qr-code-numeric-value-0.txt](https://spec.smarthealth.cards/examples/example-00-f-qr-code-numeric-value-0.txt)
* [example-00-g-qr-code-0.svg](https://spec.smarthealth.cards/examples/example-00-g-qr-code-0.svg)

Example 1[¶](https://spec.smarthealth.cards/examples/#example-1)

* [example-01-a-fhirBundle.json](https://spec.smarthealth.cards/examples/example-01-a-fhirBundle.json)
* [example-01-b-jws-payload-expanded.json](https://spec.smarthealth.cards/examples/example-01-b-jws-payload-expanded.json)
* [example-01-c-jws-payload-minified.json](https://spec.smarthealth.cards/examples/example-01-c-jws-payload-minified.json)
* [example-01-d-jws.txt](https://spec.smarthealth.cards/examples/example-01-d-jws.txt)
* [example-01-e-file.smart-health-card](https://spec.smarthealth.cards/examples/example-01-e-file.smart-health-card)
* [example-01-f-qr-code-numeric-value-0.txt](https://spec.smarthealth.cards/examples/example-01-f-qr-code-numeric-value-0.txt)
* [example-01-g-qr-code-0.svg](https://spec.smarthealth.cards/examples/example-01-g-qr-code-0.svg)

Example 2[¶](https://spec.smarthealth.cards/examples/#example-2)

* [example-02-a-fhirBundle.json](https://spec.smarthealth.cards/examples/example-02-a-fhirBundle.json)
* [example-02-b-jws-payload-expanded.json](https://spec.smarthealth.cards/examples/example-02-b-jws-payload-expanded.json)
* [example-02-c-jws-payload-minified.json](https://spec.smarthealth.cards/examples/example-02-c-jws-payload-minified.json)
* [example-02-d-jws.txt](https://spec.smarthealth.cards/examples/example-02-d-jws.txt)
* [example-02-e-file.smart-health-card](https://spec.smarthealth.cards/examples/example-02-e-file.smart-health-card)
* [example-02-f-qr-code-numeric-value-0.txt](https://spec.smarthealth.cards/examples/example-02-f-qr-code-numeric-value-0.txt)
* [example-02-f-qr-code-numeric-value-1.txt](https://spec.smarthealth.cards/examples/example-02-f-qr-code-numeric-value-1.txt)
* [example-02-f-qr-code-numeric-value-2.txt](https://spec.smarthealth.cards/examples/example-02-f-qr-code-numeric-value-2.txt)
* [example-02-g-qr-code-0.svg](https://spec.smarthealth.cards/examples/example-02-g-qr-code-0.svg)
* [example-02-g-qr-code-1.svg](https://spec.smarthealth.cards/examples/example-02-g-qr-code-1.svg)
* [example-02-g-qr-code-2.svg](https://spec.smarthealth.cards/examples/example-02-g-qr-code-2.svg)

Example 3[¶](https://spec.smarthealth.cards/examples/#example-3)

* [example-03-a-fhirBundle.json](https://spec.smarthealth.cards/examples/example-03-a-fhirBundle.json)
* [example-03-b-jws-payload-expanded.json](https://spec.smarthealth.cards/examples/example-03-b-jws-payload-expanded.json)
* [example-03-c-jws-payload-minified.json](https://spec.smarthealth.cards/examples/example-03-c-jws-payload-minified.json)
* [example-03-d-jws.txt](https://spec.smarthealth.cards/examples/example-03-d-jws.txt)
* [example-03-e-file.smart-health-card](https://spec.smarthealth.cards/examples/example-03-e-file.smart-health-card)
* [example-03-f-qr-code-numeric-value-0.txt](https://spec.smarthealth.cards/examples/example-03-f-qr-code-numeric-value-0.txt)
* [example-03-g-qr-code-0.svg](https://spec.smarthealth.cards/examples/example-03-g-qr-code-0.svg)

Changelog

Changelog[¶](https://spec.smarthealth.cards/changelog/#changelog)

1.2.1[¶](https://spec.smarthealth.cards/changelog/#121)

Documented optional exp claim for expiration

1.2.0[¶](https://spec.smarthealth.cards/changelog/#120)

Specified Health Card revocation

1.1.1[¶](https://spec.smarthealth.cards/changelog/#111)

Added verifier guidance to ignore unrecognized VC types

1.1.0[¶](https://spec.smarthealth.cards/changelog/#110)

Updated TLS requirements for issuer key set

1.0.2[¶](https://spec.smarthealth.cards/changelog/#102)

Updated links to the HL7 Implementation Guide

1.0.1[¶](https://spec.smarthealth.cards/changelog/#101)

Clarify FHIR API behavior when $health-cards-issue doesn't return results

1.0.0[¶](https://spec.smarthealth.cards/changelog/#100)

No functional change from 1.0.0-rc; added documentation links and re-worked introduction.

1.0.0-rc[¶](https://spec.smarthealth.cards/changelog/#100-rc)

No change from 0.4.5; applying tag for connectathon release

0.4.5[¶](https://spec.smarthealth.cards/changelog/#045)

Clarify mapping into VC Data Model, and strip "fixed" fields from payload

0.4.4[¶](https://spec.smarthealth.cards/changelog/#044)

Resource.meta is allowed in one special case

0.4.3[¶](https://spec.smarthealth.cards/changelog/#043)

Document CORS expectation for .well-known/jwks.json

0.4.2[¶](https://spec.smarthealth.cards/changelog/#042)

Replace iat with nbf in JWT payload encoding

0.4.1[¶](https://spec.smarthealth.cards/changelog/#041)

Added optional x5c in JWKS

0.3.1[¶](https://spec.smarthealth.cards/changelog/#031)

Add optional vcIndex param on $health-cards-issue response's resourceLink

0.3.0[¶](https://spec.smarthealth.cards/changelog/#030)

Rename $HealthWallet.issueVc to $health-cards-issue

0.2.0[¶](https://spec.smarthealth.cards/changelog/#020)

Chunk-based QR representation of larger Health Cards (JWS > 1195 characters). Defines shc:/<n>/<c>/ prefix, where <n> represents a chunk number and <c> represents the total chunk count.

0.1.1[¶](https://spec.smarthealth.cards/changelog/#011)

Added shc:/ prefix for QR representations.

0.1.0[¶](https://spec.smarthealth.cards/changelog/#010)

Significant API overhaul to reduce scope and simplify dependencies. See [PR#64](https://github.com/smart-on-fhir/health-cards/pull/64) for details.

* Remove user DIDs from the picture. They were already optional, and in some of our most important flows unlikely to be available.
* Remove the need to bind an issuer to a holder ahead of time. SMART on FHIR clients can now call $HealthWallet.issueVc without having to call $HealthWallet.connect first
* Update $HealthWallet.issueVc response to use valueString (avoids the need for base64 encoding in the FHIR Parameters resource)
* Replace DID-based key discovery with hosted JSON Web Key. Establish the requirement that Issuers host .well-known/jwks.json
* Define requirements for keeping Health Cards' JWS representation small (small enough to fit in a QR code) -- including size limits and a method for splitting a Health Card into a Health Card Set when the size limit cannot be met
* Document process for embedding Health Cards in QR codes
* Update file extension and MIME type for representing Health Cards as downloadable files (.smart-health-card and application/smart-health-card)
* Remove SIOP flow For Verifier::Holder communications

0.0.12[¶](https://spec.smarthealth.cards/changelog/#0012)

Add optional resourceLink response parameter on $HealthWallet.issueVc

0.0.11[¶](https://spec.smarthealth.cards/changelog/#0011)

Change canonical domain to https://smarthealth.cards (from https://healthwallet.cards)

0.0.10[¶](https://spec.smarthealth.cards/changelog/#0010)

Add detail on how to recognize encryption keys, signing keys, and linked domains in a DID Document

0.0.9[¶](https://spec.smarthealth.cards/changelog/#009)

Add discovery params to .well-known/smart-configuration, allowing SMART on FHIR servers to advertise Health Cards capabilities

0.0.8[¶](https://spec.smarthealth.cards/changelog/#008)

* Clarify that .fhir-backed-vc files can contain JWS- or JWE-based VCs
* Update JWS signature algorithm to ES256

0.0.7[¶](https://spec.smarthealth.cards/changelog/#007)

Simplify demographics recommendations with one uniform "minimum set"

0.0.6[¶](https://spec.smarthealth.cards/changelog/#006)

Updated encryption to use "alg": "ECDH-ES" (with "enc": "A256GCM")

0.0.5[¶](https://spec.smarthealth.cards/changelog/#005)

Updated encryption to use "enc": "A256GCM"

0.0.4[¶](https://spec.smarthealth.cards/changelog/#004)

* Added links to overview / intro video
* Updated SIOP request to identify requested credentials by type URL (https://healthwallet.cards#covid19 instead of health-wallet-covid19-card)

0.0.3[¶](https://spec.smarthealth.cards/changelog/#003)

* Update .well-known DID links and file URL to match latest spec

0.0.2[¶](https://spec.smarthealth.cards/changelog/#002)

* Use valueUri (which exists in DSTU2+) for FHIR datatypes rather than valueUrl (which was introduced after DSTU2)
* Added encryptForKeyId parameter to $HealthWallet.issueVc operation, defaulting to absent == no encryption
* Updated example VC JWT representations to ensure that the .vc.credentialSubject contains all subject-specific claims
* Defined OperationOutcome payload for failed $HealthWallet.issueVc operations