The Alan Turing Institute

Technical Brief

TinyID- QR code based verifiable credentials

January 6, 2023

Trustworthy Digital Infrastructure for Identity Systems

Dr Pamela Wochner[1]*, Dr Lydia France*[1]*, Dr Sam Greenbury*[1]*, Luke Hare*[1] *and Dr Timothy Hobson*[1]

[1] *The Alan Turing Institute*

This work was supported, in whole or in part, by the Bill & Melinda Gates Foundation [INV-001309]. The Institute is named in honour of Alan Turing, whose pioneering work in theoretical and applied mathematics, engineering and computing is considered to have laid the foundations for modern-day data science and artificial intelligence. It was established in 2015 by five founding universities and became the United Kingdom's (UK) National Institute for Data Science and Artificial Intelligence. Today, the Turing brings together academics from 13 of the UK's leading universities and hosts visiting fellows and researchers from many international centres of academic excellence. The Turing also liaises with public bodies and is supported by collaborations with major organisations.

The Alan Turing Institute
British Library
96 Euston Road
London
NW1 2DB

Contents

List of Figures

List of Tables

1 Scope of the report

The essential idea behind TinyID is to enable verifiable credentials to be shared via QR code displayed on a basic mobile device (i.e. feature phone, not smartphone). A verifiable credential (VC) is a digital document that can be used to prove a person's identity or other attributes. The World Wide Web Consortium (W3C) Verifiable Credentials Data Model and Verifiable Presentations provide flexible standards for sharing VCs [2]. However, this flexibility entails a degree of verbosity, making quick response (QR) codes an unsuitable medium for sharing VCs due to the limited data capacity of QR codes, particularly on small screens.
In this report we explore the feasibility of two alternative approaches **TinyVC** or **TinyID** for sharing VCs that support the SIMPle ID protocol (see paper [1], Section 5) with QR codes suitable for low resolution screens.

1. **TinyVC**: A verifiable credential that is i) fully compliant with W3C standard; ii) support SIMple ID protocol; iii) compressed to fit QR code; iv) compressed without loss.

   The key questions that need to be addressed are:

   - *How can the SIMple ID protocol be encoded in the W3C VC data model?*
   - *Can we achieve sufficient data compression for a low resolution QR code suitable for a basic phone?*

   The interoperability and flexibility of a TinyVC solution would be attractive to providers of digital identity services.

2. **TinyID**: A new data model and protocol derived from the W3C VC model for sharing the minimum of data that amounts to some form of verifiable presentation. TinyID would then be expanded to a TinyVC which would fit the W3C standard.

   The key questions that need to be addressed are:

   - *What is the minimum of data required that, after being shared via QR code, can be expanded into a full TinyVC?*
   - *Can we encode the TinyID data into a low resolution QR code for basic phones, and does it benefit from compression?*

   This approach sacrifices flexibility and interoperability in favour of reduced data.

2 Background

2.1 The W3C Verifiable Credential Data Model

The W3C standard for verifiable credentials (VC) is outlined in this section. Physical credentials, e.g. driver's license or passport, typically include: i) information about the type of credential; ii) information to identify the credential subject; iii) information about the issuing authority; etc. A VC can represent the same information as physical credentials, but also offer the potential to express information that have no physical equivalent. For example, a printed bank statement is not proof of ownership for that bank account, whereas a VC can include cryptographic proof and is therefore verifiable.
According to Section 4 of the W3C data model a VC must include the following properties:

- `@context`: When two software systems need to exchange data, they need to use terminology that both systems understand. This is referred to as the context of the data exchange. The value of the @context property MUST be an ordered set where the first item is a URI with the value `https://www.w3.or/2018/credentials/v1`

- `type`: Software systems that process the kinds of objects specified in this document use type information to determine whether or not a provided verifiable credential or verifiable presentation is appropriate. This specification defines a type property for the expression of type information. The value of the type property MUST be, or map to (through interpretation of the @context property), one or more URIs. If more than one URI is provided, the URIs MUST be interpreted as an unordered set.

- `issuer`: This specification defines a property for expressing the issuer of a verifiable credential. The value of the issuer property MUST be either a URI or an object containing an id property.

- `issuanceDate`: This specification defines the issuanceDate property for expressing the date and time when a credential becomes valid.

- `credentialSubject`: A verifiable credential contains claims about one or more subjects. This specification defines a `credentialSubject` property for the expression of claims about one or more subjects. The value of the property is defined as a set of objects that contain one or more properties that are each related to a subject of the verifiable credential.

- `proof`: At least one proof mechanism, and the details necessary to evaluate that proof, MUST be expressed for a credential or presentation to be a verifiable credential or verifiable presentation; that is, to be verifiable. One or more cryptographic proofs that can be used to detect tampering and verify the authorship of a credential or presentation. The specific method used for an embedded proof MUST be included using the type property.

2.2 The SIMple ID protocol

Foundational electronic ID systems commonly ask users to provide their biometrics for authentication, such as fingerprints, exposing the users to security and privacy risks. The SIMple ID paper introduces a mobile identity solution/protocol that can provide authentication credentials in the form of a QR code that is generated and displayed on a basic mobile phone. It contains a one-time password (OTP) and a resident unique identity (UID) that are required for authentication. The OTP is generated on the SIM card hardware, also called a universal integrated circuit card (UICC). For this, a one-off setup is required between the UICC and the issuer, which can be performed during manufacture. A detailed description of the full SIMple ID authentication protocol can be found in Section 5 of the SIMple ID paper. For the scope of this report, we focus on the data that is encoded in the QR code and its size.
The following table is taken from [1] and summarises the QR code capacity and sizes that can be displayed for the two most common screen resolutions of basic phones.

Version number	Capacity (bytes)	N. modules	Max module size 120×120 device (px^2)	Max module size 240×240 device (px^2)
1	17	21×21	4	8
2	32	25×25	3	7
3	53	29×29	3	6
4	78	33×33	2	5
5	106	37×37	2	5
6	134	41×41	2	4
7	154	45×47	2	4
8	192	49×49	2	4

Table 1: QR code capacity, and maximum module sizes, for the two most common basic mobile phone screen resolutions [1].

3 Methods

The example VCs used in the TinyVC and TinyID tests are described below, along with the compression and evaluation methods.

3.1 Baseline VC

First, a baseline VC that only has the properties required according to the W3C standard is considered. Such a VC can be easily obtained using `DIDKit` [3], a cross-platform toolkit that provides functionality for working with W3C VCs and W3C Decentralized Identifiers. The toolkit's `cli` were used to generate a signed example VC, shown in Listing 1, which serves as a baseline in the following feasibility study.

Listing 1: Example W3C VC generated with `DIDKit`

```
{
    "@context": "https://www.w3.org/2018/credentials/v1",
    "id": "urn:uuid:11B3400F-027C-4786-A49B-0ED1734996D5",
    "type": [
      "VerifiableCredential"
    ],
    "credentialSubject": {
      "id": "did:example:d23dd687a7dc6787646f2eb98d0"
    },
    "issuer": "did:key:
       z6MkmgSmqFySJZzbRJj4uKyqiFMw18votiZff5vYKFznZkei",
    "issuanceDate": "2022-11-16T18:02:37Z",
    "proof": {
      "type": "Ed25519Signature2018",
      "proofPurpose": "assertionMethod",
      "verificationMethod": "did:key:
          z6MkmgSmqFySJZzbRJj4uKyqiFMw18votiZff5vYKFznZkei#
          z6MkmgSmqFySJZzbRJj4uKyqiFMw18votiZff5vYKFznZkei",
      "created": "2022-11-16T18:09:29.739Z",
      "jws": "
          eyJhbGciOiJFZERTQSIsImNyaXQiOlsiYjY0Il0sImI2NCI6ZmFsc2V9..
          gxvl2W7Ni75eCi1JYf6_GRWx-
          LXWis_PxT6a4WCczKXVKMvG5T5ongu7TBH5VTVrWJcbwEi28YkIvpPbCmVhAA
          "
    }
```

```
}
```

3.2 TinyVC

The TinyVC approach is based on the minimal VC that fulfils the requirement of both the W3C VC data model and the SIMple ID protocol. The values of the VC in Listing 1 are not specific to the SIMple ID protocol and do not have a property that includes an OTP. Therefore, we modified the values of some properties to follow these specifications:

- The `issuer` of the VC is defined as `urn:<issuer>:<iso_code issuer_country>`.
- The `credentialSubject` is defined as `urn:<issuer>:<iso_ code_issuer_country>:<national_id>` and is the national ID of the credential subject and corresponds to the UID of the SIMple ID protocol.
- The `proof` property has the `type` `otp` and `proofValue` is the OTP.

Listing 2 shows the example SIMple ID VC that is used in this report. Changing the proof type to OTP significantly reduces the size of the proof property.

Listing 2: Example SIMple ID VC

```
{
  "@context": "https://www.w3.org/2018/credentials/v1",
  "type": [
      "VerifiableCredential"
  ],
  "issuer": "urn:mosip:mar",
  "issuanceDate": "2022-11-16T18:02:37Z",
  "credentialSubject": {
      "id": "urn:mosip:mar:ABCDEFGHI"
  },
  "proof": {
    "type": "otp",
    "proofValue": "0123456789"
  }
}
```

3.3 TinyID

Assuming that the values of the VC properties follow the TinyVC specifications, we can further reduce the amount of data that we need to fit into a QR code and derive the TinyID data model. This is achieved by i) discarding all "scaffolding" of the VC (such as property names); ii) discarding all redundant values that can be derived from other properties; iii) creating a standardised order of contents. However, this is only possible by sacrificing the flexibility and extensibility of the W3C VC data model. The TinyID data model makes the following assumptions:

- The `@context` must be `https://www.w3.org/2018/credentials/v1`.
- The `type` must be `VerifiableCredential`.
- The properties `id`, `issuer`, `credentialSubject` and `proof` follow the specifications in Section 3.2.
- The TinyID data model requires the data to be encoded in a standardised order (see below).

With these assumptions, it is sufficient to encode the values of `id` in the `credentialSubject` property followed by the value for `issuanceDate` (in the format `<yyyy-mm-mm>T<hh:mm:ss>Z`) and the OTP `proofValue`, separated by a comma.
This example was used:

Listing 3: Example TinyID

```
urn:mosip:mar:ABCDEFGHI,2022-11-16T18:02:37Z,0123456789
```

After having been scanned by the verifier, the TinyID data can be easily expanded into to a full VC:

- The values for the properties `@context`, `type` and `proof type` are the same for all VC within the TinyID protocol and are known by the verifier (and their verification software).
- The value for the property `issuer` can be derived from the `id` value.

This would be done by the verifier's software (outside the scope of this feasibility study), which needs to be tailored to the TinyID data model/protocol.

3.4 Compression & Evaluation

To evaluate the size of a VC, the data examples in Listings 1 - 3 were first compressed and the compressed bytes are then encoded into a QR code. The size of

the data before and after compression are computed as well as the width of the QR code. The baseline VC and TinyVC as shown in this report are formatted in a human readable way, using whitespace and newline characters to achieve this. These redundant characters are removed before compression.
These steps were implemented in Rust. For compression, `ZlibEncoder` and `Gzip` were used, which are part of the `flate2` crate [4]. `ZlibEncoder` uses the `deflate` method as compression algorithm. This is a standard lossless compression algorithm that combines Lempel–Ziv coding (LZ77) and Huffman coding [5]. `Gzip` refers to a data compression program and the resulting compressed data format. As `ZlibEncoder`, it is based on the deflate algorithm [6].
Given the sensitive nature of the data and the application, it is important that compression is lossless. Encoding the (compressed) data into a QR code was achieved with the crate `qrcode` [7].
The code is openly accessible on the TinyID Github repository [8].

4 Results

The size of the data in bytes before and after compression, as well as the size of the QR code of the baseline VC, the example TinyVC and the example TinyID for compression using `ZlibEncoder (Gzip)` are shown in Table 2. The compression ratio is obtained as the ratio (`Uncompressed size / Compressed size`).

Name VC	Size in bytes	Size compressed in bytes	Compression ratio	Width QR code
Baseline VC	705	456 (468)	1.546 (1.506)	85 (85)
TinyVC	250	193 (205)	1.295 (1.22)	57 (57)
TinyID	55	66 (78)	0.833 (0.705)	33 (37)

Table 2: Size of VCs and QR code for the example VCs for compression with **Zlib (Gzip)**.

The Figures 1 and 2 show the QR codes obtained for TinyVC and TinyID.

5 Discussion & Limitations

This report answered the following questions with **TinyVC**, a verifiable credential (VC) that is i) fully compliant with W3C standard; ii) supports SIMple ID protocol; iii) compressed to fit QR code; iv) compressed without loss.

Figure 1: QR code of the TinyVC example.

Figure 2: QR code of the TinyID example.

How can the SIMple ID protocol be encoded in the W3C VC data model?

The SIMple ID protocol requires the UID (unique ID) of a resident and an OTP (one-time password) generated by a UICC (universal integrated circuit card within a SIM card) to be encoded in a QR code. For TinyID, this information was incorporated in a W3C VC by assigning the UID as `id` value to the `credentialSubject` property and by using the OTP as `proof` value.

Can we achieve sufficient data compression for a low resolution QR code suitable for a basic phone?

Both the baseline VC and TinyVC are larger than the maximum QR code capacity for a basic phone according to Table 1. Although the data size can be reduced by approximately a factor of 1.5 for the baseline VC and by approximately a factor of 1.2 for TinyVC, the achieved compression is not sufficient to fit into a QR code suitable for display on a basic phone. The size of the `ZlibEncoder` compressed TinyVC is close to the capacity of the largest QR code listed in Table 1.

Can we encode the TinyID data into a low resolution QR code for basic phones, and does it benefit from compression?

For **TinyID**, a reduced data model was derived from TinyVC. Although TinyID itself does not represent a VC according to the W3C data model, it includes all necessary information to be expanded into a TinyVC. The TinyID data model is significantly reduced in size compared to the baseline VC and TinyVC and is small enough to fit into a QR code as specified in Table 1. The TinyID data is minimal in the sense that it does not contain redundant information and does not benefit from compression.

Future work

In this report, only the feasibility of the proposed approaches TinyVC and TinyID with respect to the data size were explored and compared to a baseline W3C VC. The results showed that only the TinyID achieves a data size that is compatible with the specifications of basic phones. The implementation of TinyID to run on such a phone is outside the scope of this report. On the user side, a fully functioning prototype for the TinyID protocol would require

- A UICC as described in SIMple ID paper that can generate the OTP.
- The required information, such as UID, to be stored on the UICC memory.
- Software that can access the generated OTP, along with the other information and generate a TinyID and encode it as QR code.

- Software to display the QR code on a basic phone.

On the verifier side, custom software is required that can process the data obtained by scanning the QR code and process it into a TinyVC.
In addition, a TinyVC W3C schema would need to be created and made openly accessible along with a detailed protocol for deriving TinyID from a TinyVC and vice versa. This would ensure that a verifiers are not dependent on any particular software but could implement their own. The schema would need to be included in the `@context` property and, consequently, the size of the corresponding TinyVC and TinyID would slightly increase.

6 Conclusions

In this report, two alternative approaches to share VCs that support the SIMple ID protocol were explored and tested. With TinyVC, a VC model was introduced that is fully compliant with the W3C data model and encodes the information required for the SIMple ID protocol. Test on an example TinyVC have shown that the size of the VC is too large to fit into a QR code that is suitible for low resolution screens. TinyID, however, is a minimal data model derived from TinyVC that fits into such a QR code without compression. In addition, TinyID can be expanded into a TinyVC

References

[1] C. Hicks, “Simple id: Qr codes for authentication using basic mobile phones in developing countries,” 2022, unpublished.

[2] “Verifiable credentials data model v1.1.” [Online]. Available: https://www.w3.org/TR/vc-data-model/

[3] “Library for verifiable credentials and decentralized identifiers.” [Online]. Available: https://crates.io/crates/didkit

[4] “A streaming compression/decompression library deflate-based streams in rust.” [Online]. Available: https://crates.io/crates/flate2

[5] “Deflate compressed data format specification version 1.3.” [Online]. Available: https://www.w3.org/Graphics/PNG/RFC-1951

[6] “Gnu gzip: General file (de)compression.” [Online]. Available: https://www.gnu.org/software/gzip/manual/gzip.html

[7] "Qr code encoder in rust." [Online]. Available: https://crates.io/crates/qrcode

[8] "Tinyid github repository." [Online]. Available: https://github.com/alan-turing-institute/tinyID