BFTKV DID Method Specification

Yahoo Paranoids
https://github.com/yahoo/bftkv

STATUS: Working Draft, September 25, 2017

The original document can be found at:
https://docs.google.com/document/d/1-FnSXk1vvJTmJ2I4Sat2MK8-7bbpKgCL2CLGGqUcqu8/edit?usp=sharing

*Note: the definitions of the terms written in **bold** are parallel with the Terminology section of the DID Specification.*

ABSTRACT OF THE DID SPECIFICATION

DIDs (decentralized identifiers) are a new type of identifier intended for verifiable digital identity that is "self-sovereign", i.e, fully under the control of the **identity owner** and not dependent on a centralized registry, identity provider, or certificate authority. DIDs resolve to **DDOs** (DID descriptor objects)—simple JSON documents that contain all the metadata needed to prove ownership and control of a DID. Specifically, a DDO contains a set of **key descriptions**—machine-readable descriptions of the identity owner's public keys—and a set of **service endpoints**—resource pointers necessary to initiate trusted interactions with the identity owner. Each DID uses a specific **DID method**, defined in a separate **DID method specification**, to define how the DID is registered, resolved, updated, and revoked on a specific distributed ledger or network.

ABSTRACT OF THIS SPECIFICATION

This document defines the BFTKV DID method specification that includes the BFTKV DID scheme and operations. Moreover, security and privacy issues are discussed in regard to DID clients, developers and owners.

BFTKV is a Byzantine fault tolerant key value storage that uses

- PGP's web of trust mechanism to build trust relationships between entities,
- b-masking quorums to provide Byzantine fault tolerance and
- Quorum certificates to restrict access to write operations

Implementation details and more information about the system is accessible at the following URL: https://www.github.com/yahoo/bftkv

Table of Contents

1. Introduction and Background

This document defines the BFTKV DID method specification adhering to the defined requirements in the DID specification.

BFTKV is a Byzantine fault tolerant key value storage that incorporates three concepts in its core:

- Byzantine Quorum Systems
- PGP's Web of Trust Mechanism
- Quorum Certificates

PGP's Web of Trust mechanism is a way to build trust between entities without a central authority. BFTKV takes advantage of this mechanism to build quorums and provide Byzantine fault tolerance. In addition, to prevent unauthorized write operations, BFTKV uses Quorum Certificates.

2. BFTKV DID Scheme

BFTKV DID scheme adheres to the following ABNF:

```
bftkv-did           = "did:bftkv:" UID
UID                 = 16*(char)
char                = ALPHA / DIGIT / '-'
```

3. BFTKV DID Operations

Prerequisites

BFTKV consists of nodes that can dynamically join and leave the system (in addition to consistently available nodes). Since the system is based on PGP's web of trust mechanism, **owner**s that will store/publish DDOs SHOULD create a PGP key pair (public and private) and use these keys for BFTKV DID operations.[1]

After the key pair is created, the keys should be marked as trusted (signed) by a quorum of the members of the BFTKV system. To achieve this, the public PGP key is sent to the system and the signatures are gathered by the client. After this step, the client can join the system and perform BFTKV DID Operations detailed below.

---

[1] In the future, BFTKV MAY provide other means of accessing BFTKV DID operations, such as via HTTP.

3.1. Create

Generate a unique identifier (UID) which can be anything as long as it is unique among BFTKV nodes. Once it is created, the UID has to be registered as a node of BFTKV. We enforce the TOFU policy. If the UID has not been registered in the BFTKV cluster the system will register the UID unconditionally and give the user back a BFTKV key pair. To recover from the key-loss situation, the user needs to prove that he or she actually owns the UID. Technically this kind of proof is out of scope of BFTKV -- for example, if the system is used as the decentralized key service for email providers, each provider is likely to provide a way to prove the email address belongs to the user. Here, however, we use the password authentication which is the default method to prove user's identity in BFTKV. The BFTKV password authentication is immune from dictionary attack even if some nodes are compromised ((k, n) threshold secret sharing scheme is used with the quorum system), and does not reveal any information about the password by eavesdropping the protocol ([SRP] is used to authenticate and to share a session key). See [BFTKV] for details.

```
% gpg --gen-key
gpg (GnuPG) 1.4.21; Copyright (C) 2015 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection?
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: My Name
```

```
Email address: foo@bar.com
Comment:
You selected this USER-ID:
    "My Name <foo@bar.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.

You don't want a passphrase - this is probably a *bad* idea!
I will do it anyway.  You can change your passphrase at any time,
using this program with the option "--edit-key".

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++++
.......+++++
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
............+++++
...+++++
```

Then create a BFTKV instance with the path to the public and secret key rings and register it with the password:

```
bftkv := Bftkv.New(".gnupg")    // create an instance for all
operations
bftkv.Register("password")
```

When the registration process has finished, the public key ring has signatures from a quorum.

Now the user can construct a DID and DDO by themselves following the DID specification. UID is will be the key to associate DID/DDO with the BFTKV node.

BFTKV does not care about the content of DDO. When the user has created <DID, DDO> he can “write” it to the BFTKV cluster using a BFTKV API:

```
bftkv.Write(DID, DDO)
```

If DDO includes private data or the user does not want to propagate data in plain they might want to consider to use a simple private directory service to keep DDO. In this case BFTKV provides data integrity only.

```
bftkv.Write(DID, H(DDO))
```

Users can check the hash value after retrieving the actual DDO from the private service.

Another possibility is to use the password roaming service of BFTKV. The cluster can keep the passwords associated with variable (DID in this case). A user encrypts DDO with a password and simply writes the encrypted data:

```
bftkv.WriteWithPassword(DID, DDO, <password>)
```

Then the user can retrieve the data from anywhere simply using the password. The password is protected by the quorum the same way as the TOFU password.
`DDO := bftkv.ReadWithPassword(DID, <password>)`
Needless to say, the reader and writer need to share the password.

Data Integrity

BFTKV guarantees the data integrity. The data returned from the API is guaranteed not to be tampered unless the whole system is compromised. (See [BFTKV] security analysis for the threshold.) If users need additional data integrity (it can be considered as the application layer data integrity), the signature field of DDO can be used for that purpose and BFTKV provide APIs to sign and verify. Canonicalization of DDO is out of scope of BFTKV -- it signs/verifies data as-is whatever it is. The signing key is the PGP key generated by the user. Unlike ordinary signature scheme, BFTKV uses the quorum certificate to verify the signature. With this scheme, users can verify the DDO signature without specifying any certificate of the signer or other authorities. The quorum system collaboratively checks the signature.

```
tbs := Canonicalize(DDO)   // should not contain the signature part
sig := bftkv.Sign(tbs)
encodedSig := encode(sig)
// construct the complete DDO

tbs := Canonicalize(DDO)
encodedSig := parse(DDO).Signature.Data
sig := decode(encodedSig)
verified := bftkv.Verify(tbs, sig)
```

Proof of Control

BFTKV has its own permission system to update data associated with keys, and its own way to recover from the key-loss situation. Therefore we do not require “proof of control”.

3.2. Read/Verify

A BFTKV DID record can be looked up directly by the DID using a standard BFTKV API that simply takes the DID and will return the DDO.

```
DDO := bftkv.Read(DID)
```

BFTKV guarantees data integrity and freshness of the value.

3.3. Update

BFTKV DID records can be updated directly using a standard BFTKV API that simply takes the DID and the updated DDO.

```
bftkv.Write(DID, DDO)
```

BFTKV guarantees only the original writer of DDI can update the same DDI with a new value (updated DDO). See [BFTKV] for details.

3.4. Delete/Revoke

Delete/Revoke operation for a DID can be achieved by writing null to DDO.

4. Security Considerations

4.1. Transport security

BFTKV uses PGP message encryption scheme to protect the message from: eavesdropping, replay, message insertion, deletion, modification, and man-in-the-middle attacks.

4.2. Potential denial of service attacks

The transport security mitigates DoS type attacks to some extent. Also, TOFU (Trust On First Use) will prevent massive unauthorized writes before it goes to the storage.

4.3. Residual Risks

The quorum system mitigate errors at some nodes. See [BFTKV] for details.

4.4. Integrity protection and update authentication

BFTKV has two update modes: TOFU and WriteOnce. With the TOFU policy, the key/value slot can be modified by the original writer. BFTKV uses quorum certificate to recover from the key-loss situation. See [BFTKV] for details.

4.5. Burdens

Known as “load” in the quorum system. BFTKV uses two separated quorum systems to minimize the load at each node: authority and r/w quorums. See [BFTKV] for details.

4.6. Uniqueness of DID

It is the user’s responsibility to choose the unique ID as the DID method specific part. BFTKV can accept any unique ID and links to DID/DDO.

4.7. Private key management

BFTKV uses “GPG secring” to keep private keys. Both pubring and secring MUST be securely stored at each BFTKV node. **Owners** are responsible for keeping the keys secure.

4.8. Identity Owner Authentication and Verifiable Claims

See 3.1 Create.

5. Privacy Considerations

BFTKV DID with its Security Considerations, aims to provide a usable and secure environment for DIDs, owners and other system entities. In addition to this, BFTKV DID does not store any personally-identifiable information. However, DID owners might need to take precautions to reduce the correlation risks. These include sharing a private DID for each relationship, providing unique information in each DDO, as suggested by the DID Specification.

References

[BFTKV] A Byzantine Fault-Tolerant Key Value Storage (https://github.com/yahoo/bftkv/blob/master/docs/bftkv.pdf)
[SRP] The SRP Authentication and Key Exchange System (https://www.ietf.org/rfc/rfc2945.txt)