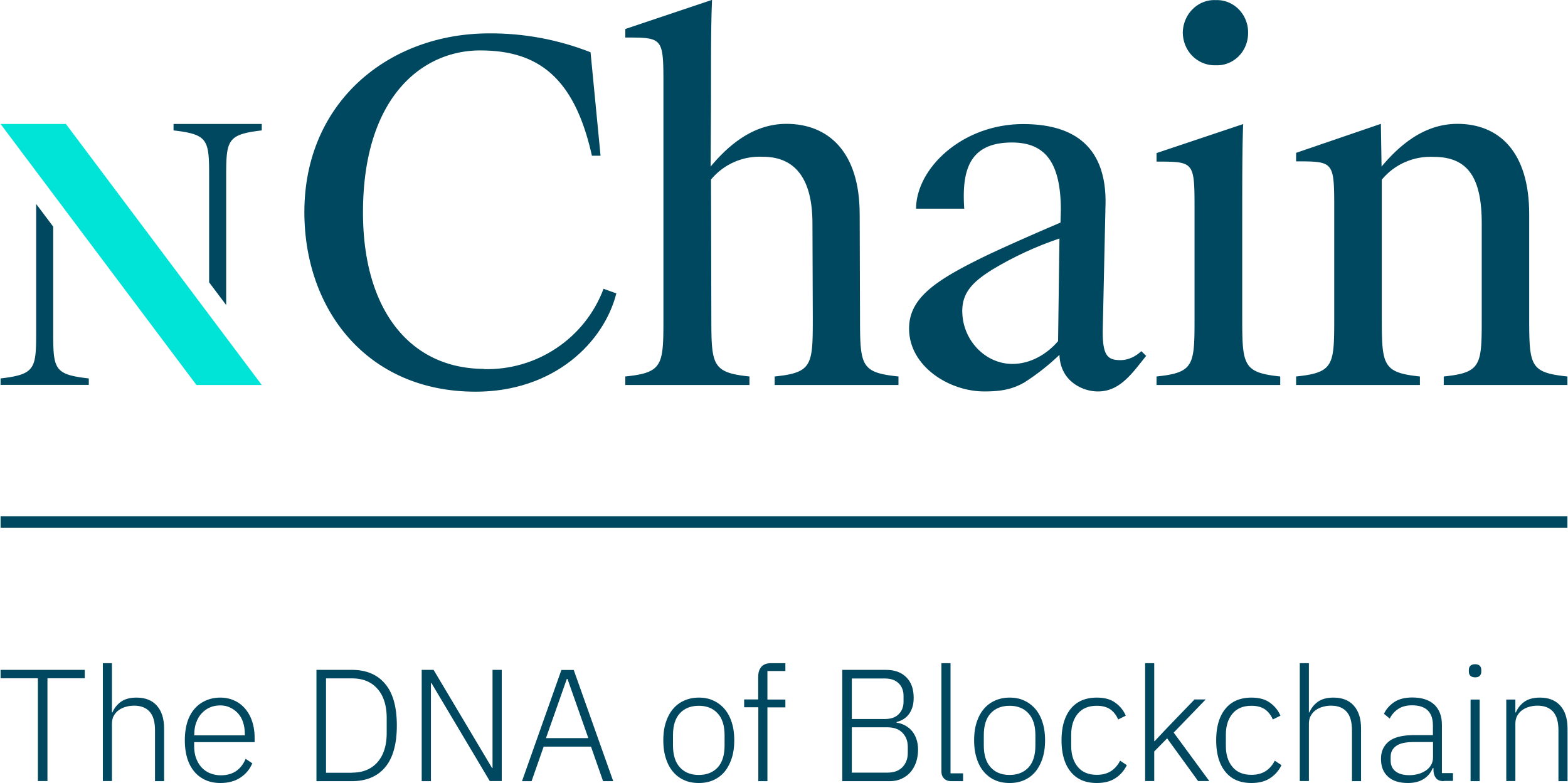
1Background pattern

Description automatically generatedBackground pattern

Description automatically generated



UTXO DID Method

Version A07

Created 25/09/2024

Last updated 26/09/2024

## Change Log

### Revision History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Summary of changes** |
| A01 | 21/05/2024 | Initial Draft |
| A02 | 24/06/2024 | First Iteration Thomas |
| A03 | 31/06/2024 | Review and improvement |
| A03 | 05/07/2024 | Second iteration with Thomas |
| A03 | 12/07/2024 | Review from Marko Macek |
| A04 | 22/07/2024 | Changes applied |
| A04 | 24/07/2024 | Changes applied |
| A04 | 25/07/2024 | Leon Mlakar |
| A05 | 28/07/2024 | Huge Lopez |
| A06 | 13/08/2024 | Owen Vaughan and Liuxuan Pan |
| A07 | 26/09/2024 | Leon Mlakar, Božo Dragojevič |

### Production

|  |  |  |
| --- | --- | --- |
| **Name** | **Title** | **Contact** |
| Thomas Moretti | Head of Product | [t.moretti@nchain.com](mailto:t.moretti@nchain.com) |
| Taz Uddin | Senior Technology Consultant | [t.uddin@nchain.com](mailto:t.uddin@nchain.com) |
| M. Eugenia Lopez | Technical Product Lead – Identity | [h.lopez@nchain.com](mailto:h.lopez@nchain.com) |

### Distribution

Document maturity A - Draft | B - Revision | C - Final | D - Approved | E - Archived | F - Dispose

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Issued to** | **Contact** | **A01** | **B01** | **C01** | **D01** | **E01** | **F01** |
| Maria Eugenia Lopez | h.lopez@nchain.com |  |  |  |  |  |  |
| Thomas Moretti | t.moretti@nchain.com |  |  |  |  |  |  |

## Contents

[Change Log 1](#_Toc178329295)

[Revision History 1](#_Toc178329296)

[Production 1](#_Toc178329297)

[Distribution 2](#_Toc178329298)

[Contents 3](#_Toc178329299)

[1. Abstract 4](#_Toc178329300)

[2. Introduction 4](#_Toc178329301)

[2.1 Purpose 4](#_Toc178329302)

[2.2 Intended audience 4](#_Toc178329303)

[2.3 Scope 4](#_Toc178329304)

[2.4 Compliance 5](#_Toc178329305)

[2.5 Terminology 5](#_Toc178329306)

[3. Specification Overview 7](#_Toc178329307)

[3.1 Prerequisites 7](#_Toc178329308)

[3.2 UTXO DID Method 7](#_Toc178329309)

[3.3 DID Controller and DID Resolver 8](#_Toc178329310)

[3.4 DID Operations 10](#_Toc178329311)

[4. Governance model using DID BSV method. 18](#_Toc178329312)

[5.Low latency DID resolution 19](#_Toc178329313)

[6. Data register analysis 20](#_Toc178329314)

[7. Privacy Considerations 21](#_Toc178329315)

[A. UTXO DID Method Normative Reference 22](#_Toc178329344)

[A.1 DID Syntax 22](#_Toc178329345)

[A.2 Verifiable Data Registry 22](#_Toc178329346)

[A.3 DID Operations 25](#_Toc178329347)

[References and Reading Material 28](#_Toc178329348)

1. Abstract  
This document outlines a method for the issuance, status updates, key rotation, and revocation of Decentralized Identifiers (DIDs) using a public, UTXO-based blockchain as a data registry. In this approach, DIDs are issued and managed through transactions recorded on the blockchain, leveraging its transparency and immutability. The blockchain allows a UTXO to be spent only once, which provides a double spend protection, resulting in a final spent state that is permanent and verified by the network. This characteristic makes the UTXO model ideal for reflecting the status of a DID, such as revocation, which, is irreversible once executed. DIDs can also be updated by linking a transaction chain to the DID Document, with a new version controlled solely by the DID Controller. Our approach supports both issuer-initiated and user-initiated status updates and revocation requests, ensuring secure and verifiable DID management within a distributed ecosystem. This document describes key rotation methods for DID recovery. We described a proposal to accomplish low latency for DID resolution and a governance mechanisms to guide the implementation and deployment of DIDs, emphasizing the critical role of the blockchain in maintaining a reliable and transparent data registry.

## 2. Introduction

### 2.1 Purpose

The purpose of this documentation is to specify a DID method for DID issuance and management. This method is aimed to follow W3C specifications. The main differentiator is to feature an enhanced DID Document status update and revocation method that can be implemented in any public, UTXO blockchain used as the data register. The advantages are a follow.

* The double spend protection of blockchain means that transaction chains provide an immutable, unique, timestamped sequence of events. We use this to record DID issuance, status update, key rotation, and revocation. UTXO-based blockchains support parallelisation of transaction validation which makes them scalable.
* The digital signatures already present in blockchain transactions are used to provide the DID authorisation system. It supports hierarchal public key infrastructure that establishes governance and hierarchical authorisation over DID issuance.
* The programmability of blockchain transactions means that multi-party authorisation schemes can be easily implemented. For example, by allowing either a DID Controller or a DID Subject to revoke a DID using a 1-of-2 multi-signature scheme in a transaction locking script.

### 2.2 Intended audience

This specification is intended for software implementers that want to adopt this method for the creation and verification of Decentralized Identifiers. The implementation of this method was chosen to be on the Bitcoin SV blockchain due to its low transaction fees, high throughput, and instant transaction verification. But it is understood that it can be implemented in any UTXO-based blockchain. These specifications assume a basic understanding of programming and blockchain technology.

### 2.3 Scope

The specifications include:

* The use of a public blockchain as a data registry.
* Details and specifications on how to link the public key to a blockchain transaction (Tx).
* Transaction anatomy and a description of the signature structure: a multi-signature for DID issuance and DID Document creation. These signatures are coordinated to the DID Subject and the DID Controller by a component service that creates and manages the DID called a *DID manager* which is run by an authorized entity. DID issuance can be triggered by any entity, or any wallet connected with the DID manager. The specifications also explain how to link a blockchain transaction to a status check from the DID manager.

This design allows DID status verification to be performed independently of the DID issuer and the DID manager which provides privacy enhancement, as the DID issuer and the DID manager are not aware of the status checks being performed by the verifier. Consequently, users can verify the status of their DIDs without revealing their actions to the issuers, which also increases the privacy in the verification process. This method can be applied universally across different UTXO-based blockchain networks, making it a versatile solution for decentralized identity management.

### 2.4 Compliance

Our implementation of the DID method can appropriately handle and enforce the issuance, status check revocation, and verification status of the DID and DID Document in accordance with the specifications presented by the[W3C](https://www.w3.org/TR/did-core/#sotd), and the [DIF](https://decentralized-id.com/).

### 2.5 Terminology

The following terms are used to describe concepts in this specification.

|  |  |
| --- | --- |
| **Terms** | **Description** |
| Blockchain | The Bitcoin blockchain is the transaction database that is extended by nodes participating in the [mining](https://wiki.bitcoinsv.io/index.php/Mining) process on the Bitcoin network. The blockchain contains [transactions](https://wiki.bitcoinsv.io/index.php/Transactions) validated and processed by nodes on the network. Using the blockchain, anyone can add and verify a record of all of the [confirmed](https://wiki.bitcoinsv.io/index.php/Confirmation) transactions that have taken place on the ledger up until the most recent block was found.  ([BSV Blockchain, UTXO](https://wiki.bitcoinsv.io/index.php/Blockchain)) |
| Blockchain Transactions | A Bitcoin transaction consists of a version number, a lock time value, a list of inputs and a list of outputs. The primary functionality of a Bitcoin transaction is to transfer custody of bitcoin from one person to another. A transaction uses [UTXOs](https://wiki.bitcoinsv.io/index.php/UTXO) as inputs and distributes their value to new outputs. UTXOs are the 'coins' in which all bitcoins are stored.  ([BSV Wiki, Bitcoin Transactions](https://wiki.bitcoinsv.io/index.php/Bitcoin_Transactions)) |
| Chain of Dust | In Bitcoin Satoshi Vision (BSV), a "Chain of Dust" refers to a sequence of minimum required value transactions that are created by spending small amounts of Satoshi’s. This concept is related to "dust" in the cryptocurrency context. **These** are small blockchain transactions where one dust transaction leads to another. Each subsequent transaction in the chain is dependent on the previous one, creating a linked sequence.  ([BSV Wiki, UTXO](https://wiki.bitcoinsv.io/index.php/UTXO)) |
| [Decentralised identifier](https://www.w3.org/TR/did-core/#dfn-decentralized-identifiers) | A globally unique persistent identifier that does not require a centralized registration authority and is often generated and/or registered cryptographically. The generic format of a DID is defined in [3.1 DID Syntax](https://www.w3.org/TR/did-core/#did-syntax). A specific [DID scheme](https://www.w3.org/TR/did-core/#dfn-did-schemes) is defined in a [DID method](https://www.w3.org/TR/did-core/#dfn-did-methods) specification. Many—but not all—DID methods make use of [distributed ledger technology](https://www.w3.org/TR/did-core/#dfn-distributed-ledger-technology) (DLT) or some other form of decentralized network.  [(W3C. DIDs v1.0, 2022)](https://www.w3.org/TR/did-core/#dfn-did-documents) |
| DID Controller | An entity that has the capability to make changes to a [DID document](https://www.w3.org/TR/did-core/#dfn-did-documents). A [DID](https://www.w3.org/TR/did-core/#dfn-decentralized-identifiers) might have more than one DID controller. The DID controller(s) can be denoted by the optional controller property at the top level of the [DID document](https://www.w3.org/TR/did-core/#dfn-did-documents). Note that a DID controller might be the [DID subject](https://www.w3.org/TR/did-core/#dfn-did-subjects).  [(W3C, DID V.1.0,2022)](https://www.w3.org/TR/did-core/#dfn-did-controllers) |
| DID Document | A set of data describing the [DID subject](https://www.w3.org/TR/did-core/#dfn-did-subjects), including mechanisms, such as cryptographic public keys, that the [DID subject](https://www.w3.org/TR/did-core/#dfn-did-subjects) or a [DID delegate](https://www.w3.org/TR/did-core/#dfn-did-delegate) can use to [authenticate](https://www.w3.org/TR/did-core/#dfn-authenticated) itself and prove its association with the [DID](https://www.w3.org/TR/did-core/#dfn-decentralized-identifiers). A DID document might have one or more different [representations](https://www.w3.org/TR/did-core/#dfn-representations).  [(W3C, DID V.1.0,2022)](https://www.w3.org/TR/did-core/#dfn-did-controllers) |
| DID [Issuer](https://www.w3.org/TR/vc-data-model-2.0/#dfn-issuers) | An individual or entity that issued a DID. This term is specific to this document. |
| DID Manager | In this document we call DID manager to a package that includes the DID resolver and DID controller. DID Manager term is specific to nChain implementation. |
| DID Resolver | A [DID resolver](https://www.w3.org/TR/did-core/#dfn-did-resolvers) is a software and/or hardware component that performs the [DID resolution](https://www.w3.org/TR/did-core/#dfn-did-resolution) function by taking a [DID](https://www.w3.org/TR/did-core/#dfn-decentralized-identifiers) as input and producing a conforming [DID document](https://www.w3.org/TR/did-core/#dfn-did-documents) as output.  [(W3C, DID V.1.0,2022)](https://www.w3.org/TR/did-core/) |
| [DID Subject](https://www.w3.org/TR/vc-data-model-2.0/#dfn-holders) | The entity identified by a [DID](https://www.w3.org/TR/did-core/#dfn-decentralized-identifiers) and described by a [DID document](https://www.w3.org/TR/did-core/#dfn-did-documents). Anything can be a DID subject: person, group, organization, physical object, digital thing, logical thing, etc. [(W3C,DID V.1.0, 2022, Section 5.1.1)](https://www.w3.org/TR/did-core/#dfn-did-documents) |
| [Ledger](https://www.iso.org/obp/ui/en/#iso:std:iso:22739:ed-2:v1:en) | In this document, the ledger is distinguished from the blockchain. The Bitcoin ledger refers to information store that [keeps records (3.81)](https://www.iso.org/obp/ui/en/#iso:std:iso:22739:ed-2:v1:en:term:3.81) of [transactions (3.93)](https://www.iso.org/obp/ui/en/#iso:std:iso:22739:ed-2:v1:en:term:3.93) that are intended to be final, definitive and [immutable (3.51)](https://www.iso.org/obp/ui/en/#iso:std:iso:22739:ed-2:v1:en:term:3.51). Unlike the blockchain, which is a chain of blocks verified by miners, the ledger specifically focuses on the individual transactions.  ([SO22739-2024](https://www.iso.org/obp/ui/en/#iso:std:iso:22739:ed-2:v1:en)) |
| UTXO | UTXO is an acronym for unspent transaction output. Every [Bitcoin transaction](https://wiki.bitcoinsv.io/index.php/Bitcoin_Transactions) in every [block](https://wiki.bitcoinsv.io/index.php/Block) contains at least one [output](https://wiki.bitcoinsv.io/index.php/VOUT). Outputs are then spent by inputs of later blockchain transactions and typically must be unlocked with a [digital signature](https://wiki.bitcoinsv.io/index.php/Digital_Signatures_in_Bitcoin) (most commonly [ECDSA](https://wiki.bitcoinsv.io/index.php/ECDSA) in Bitcoin). Until an output is used as an [input](https://wiki.bitcoinsv.io/index.php/Input) in another transaction, this output is called a UTXO.  ([BSV Wiki, UTXO](https://wiki.bitcoinsv.io/index.php/UTXO)) |
| [Verifiable data registry](https://www.w3.org/TR/vc-data-model-2.0/#dfn-verifiable-data-registries) | In order to be resolvable to [DID documents](https://www.w3.org/TR/did-core/#dfn-did-documents), [DIDs](https://www.w3.org/TR/did-core/#dfn-decentralized-identifiers) are typically recorded on an underlying system or network of some kind. Regardless of the specific technology used, any such system that supports recording [DIDs](https://www.w3.org/TR/did-core/#dfn-decentralized-identifiers) and returning data necessary to produce [DID documents](https://www.w3.org/TR/did-core/#dfn-did-documents) is called a [verifiable data registry](https://www.w3.org/TR/did-core/#dfn-verifiable-data-registry). Examples include [distributed ledgers](https://www.w3.org/TR/did-core/#dfn-distributed-ledger-technology), decentralized file systems, databases of any kind, peer-to-peer networks, and other forms of trusted data storage.  (W3C, DID V.1.0,2022) |
| [Verifier](https://www.w3.org/TR/vc-data-model-2.0/#dfn-verifier)s | In this document, a verifier refers to an entity, organization, or individual that checks the status of the DID and verifies that the signature of the subject or DID controller matches the signatures in the presentation.  (W3C, DID V.1.0,2022) |

## 3. Specification Overview

THIS SECTION IS NOT NORMATIVE

### 3.1 Prerequisites

According to [W3C specifications](https://www.w3.org/TR/did-core/) and as a requirement for the deigns of this method, the DID needs to fulfil the basic premise principles listed below:

* A Permanent identifier (URN).
* Be cryptographically verifiable.
* Be decentralised (no central registration authority required).
* Easy and economical to create.
* The DID is always required to resolve (associated with) to a DID Document.
* The DID Subject is not always the administrator of its DID. It depends on use case scenario.

Similarly, the DID Document needs to satisfy the following:

* The DID Document needs to be publicly available.
* The DID Document must contain:
  + One or more keys to authenticate the DID subject.
  + One or more Services associated with DID subject.
  + Additional metadata like digital signatures, timestamps, and other cryptographic proofs.

### 3.2 UTXO DID Method

In this section we present an overview of the DID method. Our proposal introduces a new Decentralized Identifier (DID) method that links a UTXO and the DID with the public key of the subject and manages the DID status and revocation through transaction spending checks. We use the BSV blockchain as the verifiable data registry.

The UTXO DID method uses the following method name: **bsv**

##### The DID

The issuance of a DID using this method is performed by publishing a blockchain transaction: ***Tx0.***This transaction has a single input and a single output. The transaction ID (***TxID)***becomes the DID for the subject. The ***TxID*** can be easily calculated by either controller or subject and, because of the properties of blockchain it would never be repeated. It is also

easily recognisable by independent third-party platforms, such as blockchain explorers.

A screenshot of a number

Description automatically generated

Figure 1. Representation of a DID as per W3C specification.

*Figure 2. Representation of a DID using UTXO method.*

A number on a white background

Description automatically generated

##### The DID Document

The DID Document is published via a subsequent transaction ***Tx1*** that spends the output of ***Tx0***.The relationship between the DID, the DID document, and the blockchain transactions is given in *Figure 3.* The transaction ***Tx1*** contains a single input and a single output. The output contains the locking script, the DID Document and the funds covering the mining fee of the next transaction. ***Tx1*** spending the output of ***Tx0*** allows an external observer to conclude that there is a link between both blockchain transactions. The status of A screenshot of a computer

Description automatically generated***Tx1*** output indicates the latest status of the DID Document.

Figure 3

##### The DID Document Data Model

Our current implementation uses [W3C DID Document Data Model](https://www.w3.org/TR/did-core/#data-model) and is referenced in the DID Document. Below is a representative example of a DID Document data model.

"@context": "<https://www.w3.org/ns/did/v1>",

"id": "did:bsv:5909468ac49f960e191faba2dd7da60bd1775ccf59e90b8390c971d04741b710",

"controller": "did:bsv:b6333300b727ae4d355ffe2fee06ebe9ed5565cb1321e02637fa971bd523272e",

"verificationMethod":

[ {

"id": "did:bsv:5909468ac49f960e191faba2dd7da60bd1775ccf59e90b8390c971d04741b710#subject-key",

"type": "JsonWebKey2020",

"controller": "did:bsv:5909468ac49f960e191faba2dd7da60bd1775ccf59e90b8390c971d04741b710",

"publicKeyJwk":

{

"kty": "EC",

"x": "xJxFTwL183Hmz19WnLAgBa1wpljMuaYk\_rBnAKlol-g",

"y": "585wM9i1dGHr6qgL5NG5N2EAxel3Son9HpkGpl-hY-I",

"crv": "secp256k1"

} } ]

"authentication": [ "did:bsv:5909468ac49f960e191faba2dd7da60bd1775ccf59e90b8390c971d04741b710#subject-key" ] }

### 3.3 DID Controller and DID Resolver

*THIS SECTION IS NOT NORMATIVE*

##### DID Controller

As described in W3C, the DID Controller is an entity that has the capability to make changes to a [DID document](https://www.w3.org/TR/did-core/#dfn-did-documents). A DID Controller is not a central registration authority, subjects can be identified by multiple DIDs issued by different DID Controllers. Initially the DID Controller is required to create an identity for itself. The process describe here is the same required for an external Subject DID Creation: the DID controller requires an issuance transaction ***Tx0’***, locked to its own public key PKC0 and the “subject” public key PKCD. In this case, the subject is the controller itself. The transaction ID of ***Tx0’***, ***TxID0’*** becomes the DID of the Controller. To generate the subsequent transaction that will contain the Controllers DID Document, the controller will sign the transaction using two public keys: PKCD and PKC0, both of which belong to the Controller. ***Tx1’*** input spends the output of ***Tx0’***.

When requested by the subject, the DID Controller issues a new DID by creating and broadcasting two blockchain transactions: an Issuance and DID Document transaction. The issuance transaction, ***Tx0*** is created by the DID Controller and provisioned (funded) by controller. UTXO of ***Tx0*** should provide enough funds to cover the mining fee for the next transaction. The transaction ID of ***Tx0***, ***TxID0*** becomes the DID of the Subject. The DID Document is contained in the subsequent transaction ***Tx1,*** linked to the issuance transaction ***Tx0*** by spending its output. ***Tx1*** has a single output; that contains a payload with the DID Document and the minimal required funding to cover the mining fee pf the next transaction.

##### DID Resolver

As described by the W3C, the DID resolver will resolve the DID to a DID Document. To read the DID Document, the DID resolver will search for the corresponding transactions in the ledger.

The DID contains the transaction ID of the issuance transaction, ***Tx0***. The transaction spending its output, ***Tx1***, contains the first version of the DID Document. The most recent DID document is the document contained in the last transaction in the chain, ***Tx****n*, of which UTXO is not spent.

The DID document *versionId* is the transaction ID of the transaction containing it.

The DID document *versionTime* is the timestamp of the blockchain block containing the transaction containing the DID it.

A diagram of a system

Description automatically generated

The verifier uses the DID method specific part of the DID identifier to search for the transaction in the ledger and find the linked transactions that contain the DID Document to perform status checks on their outputs.

The verifier can request DID status using a DID resolver. Verifiers can build and run an independent DID resolver, create their own implementation (abiding by this specification), or use a service provided by a third party that runs an implementation of the UTXO DID method.

Figure 4

### 3.4 DID Operations

In this section, we will cover specifications to Create, Update, Read, and Revoke a DID. We included implementation examples as references. Please note that some specifications are optional and serve only to illustrate potential implementations.

#### 3.4.1 Create

In previous sections, we have explained how the DID and DID document are related through two transactions, and how the Controller issues a DID to itself. This section explains in detail the process of a DID Controller creating a DID and a DID Document for Subject, including the digital signatures and the transaction specification required. It also explains the use of signatures in blockchain transactions to bind the Subject and the Controller public keys to the DID and the DID Document.

Recall that the DID Controller possesses two keys:

* PKCD. Which serves as the Controller’s master key. It is only used to establish the Controllers DID.
* PKC0. Which is used to create/sign DIDs for DID subjects.

The Subject has a single key.

* PKS0. This key is always in possession of the subject.

In order for the DID Controller to issue a DID to a Subject the following transactions must be:

**TxID0: Issuance Transaction**.  
When a new DID is requested by the subject, the Controller will create an issuance transaction. This transaction locks the output to two public keys: the Controller's public key and the subject's public key. This initial transaction is necessary to initiate the DID Document process. Its transaction IDis used in the DID string: ***did:bsv:TxID0*** as the Subject’s DID***.***

***Tx0*** has a single input and a single output. The DID Document is published via a subsequent transaction ***Tx1*** that spends the output of ***Tx0***. *See Figure 5.* Note that when the Controller is required to issue a DID for itself, the Issuance transaction locks the output to both controller’s public keys (PKCD and PKC0). *See Section 3.3. DID Controller.*

**TxID1: DID Document transaction**   
This is a subsequent transaction that spends the output ofthe issuance transaction ***Tx0***. This transaction has a single input and a single output. This output contains a data payload with the DID Document. ***Tx1*** output is locked to a one-of-two multi-sig script signed either by Subject public key or the Controller public key. Once the input of ***Tx1*** is signed by both keys, the transaction is broadcasted to the network. When validated by miners it will read as resolved by the DID Resolver.

A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

Figure 6

Figure 5

***Data payloads in the transaction outputs*.**

After OP\_RETURN we can find a data payload. This data does not affect the spending conditions, and it is described below:

* **<BSV DID>**. Identifier specifying this is a transaction associated with the BSV DID Method.
* **<Identity Code>.** DID Controller configuration identifier.
* **<*DID Document JSON*>** DID Document *or* type transaction indicator **<1/2/3>:** Issuance, Update or key rotation and Revoke respectively.

##### Implementation Example

Below we provide a detailed implementation of the BSV DID Method as a reference example for DID issuance. This specification covers the process of requesting a DID from the subject to the DID controller. It includes two specific examples of the implementation:

* **DID Controller Role:** The DID controller can be run by any entity. It is important to note that the BSV DID method does not require a centralized controller. Any organization or entity can run a DID controller and create a DID for itself, as specified in *Section 3.3*.
* A screenshot of a computer

  Description automatically generated**Database Persistence:** We demonstrate database persistence in this implementation as an example, although this is optional based on implementation preferences.

Figure 7

**Key Considerations**

* **Proving Ownership**: The DID Subject demonstrates ownership of their private key by signing transaction Tx1 and sending it to the DID Controller.
* **Transaction Tx0**: The output of ***<Tx0>*** is locked by the public keys of both the DID Controller and the DID Subject.
* **Transaction Tx1**: The input of ***<Tx1>*** requires signatures from both the Subject and the Controller. The output will contain the Controller-generated DID Document. This output is locked using a one-of-two multi-signature, allowing either the Controller or the Subject to spend the output.
* **Contents of the DID Document:**
  + DID of the Subject.
  + Public Key associated with the DID Subject.
  + DID of the Controller (this identifies the DID Controller by resolving their DID).
  + Authentication method for the DID Controller (public key that can be verified on the signature input signing ***Tx1***)***.***

#### 3.4.2 Resolve

The resolver uses the DID (TxID) to find the transaction in the ledger and identify the linked transactions containing the DID Document. Once the DID Document Tx has been identified the resolver does a status check on the output of the Tx via the DID Resolver. Who will find the TxID given and track UTXO status until the last UTXO update is found.

The DID Resolver can be independently built and operated, implemented according to this specification, or provided by a third party that supports the UTXO-DID method. This method permits the use of different resolvers, ensuring the users are not required to rely on or trust our specific implementation.

#### 3.4.3 Update

##### A. DID Document Status Update

A screenshot of a computer

Description automatically generatedIn this section we explain how to change the status of the DID Document. The DID controller can change or update the status of the DID Document by creating a new transaction that spends the output of the latest transaction referring to the DID Document status. *See Figure 8*.

Figure 8

##### B. Rotation of Keys

Both subject and controller can initiate rotation of keys. How the rotation is performed depends on who is initiating and for what reason. Scheduled or voluntary rotations are different from rotations forced because keys have been compromised.

###### **Below we listed a series of use cases:**

* User Key Rotation with Valid Subject's Key
* User Key Rotation upon Compromised Subject's Key
* User Key Rotation because controller rotated its <PKC0> N key (user does not know if N was compromised or rotated regularly)
* Controller rotation of valid or compromised M key <PKCD>
* Controller rotation to update a valid <PKC0> key
* Controller rotation upon <PKC0> key compromised

Subject Key Rotation

There are many reasons a Subject key needs to be rotated: either regular operating reasons like voluntary, scheduled rotation to comply with policies, security reason like a key being compromised or to keep a valid controller attestation because the controller key has been rotated.

* Subject Key’s compromised: In case the subject’s key was compromised, it cannot be used to attest the rights, and the DID controller will have to establish some authentication process of the subject that does not refer to the compromised keys. Once the DID Subject is authenticated, the DID controller will discover that “PKS0” has been compromised and will sign a new issuance transaction using its PKC0 key. The output spent from the transaction referring to the latest DID Document transaction: ***TXn*** becomes the input of ***TXf (funding Tx).*** This transaction locked the new output to the controller’s unchanged public key and the subject’s new public key.
* Subject Key’s not compromised: In all other cases the authentication of the subject via an external channel is not necessary as the old key is considered secure and DID Subject will have to use both old and new keys to attesting that it is authorized to rotate the key as well as prove that it owns the new key.

A screenshot of a computer

Description automatically generated

Controller Key Rotation

Figure 9. Notification of key rotation of end-user is a user experience decision, configurable and not mandatory. Invalid signatures will be rejected by blockchain.

There are many reasons a Controller key needs to be rotated: either regular operating reasons like voluntary, scheduled rotation to comply with policies, security reason like any of the Controller keys: PKCD or PKC0 has been compromised. The Controller can lose any of the following keys.

* <PKCD>: Master Key which changes their own DID Authentication as well as all the issued DIDs.
* <PKC0>: Public Key associated with their DID document, i.e. the Root key for all their DID Issuance.

Only in the scenario in which the Controller losses his master key ***<PKCD’>,*** would require a different process for Key rotation. When changing ***<PKC0>*** the Controller will just be able to update the keys themselves.

Compromised PKCD: The process begins with the DID Controller performing a Master Key update, during which the Controller updated its key to the new Master Public Key ***<PKCD’>***. Subsequently, the controller generates a new DID for itself ***<TxID0’>*** , linking the previous public key to the new one, and then creates and publishes a new DID Document ***<TxID1’>*** spending ***<TxID0’>*** unique output.

A screenshot of a computer

Description automatically generatedAs a result of the controller keys being updated, the DID Document of all subjects that were co-signed by the Controller Key’s needs to be updated. In order to do so, the Controller will be required the DID Document controller keys of all subjects to be rotated. The DID Subject accepts the key rotation request. The DID Controller then generates a Key Rotation transaction, that the Subject must sign to spend the output of the ***TxIDn*** that contains the last status of the DID Document.

A new DID issuance and DID Document transaction is funded using a new Master Key. Afterwards, the Rotation Transaction is sent to the DID Subject for signing. Once signed, it is returned to the controller. The Rotation Transaction is then stored and broadcast on the blockchain. Following this, a new DID Document for the user is generated, incorporating a new derivation factor and updated keys. A Publish Transaction is created for the DID Subject, which is also signed and returned after signing. This Publish Transaction is then stored and broadcast on the blockchain. Finally, the updated DID Document is confirmed to have been successfully created and recorded on the blockchain.

A screenshot of a computer screen

Description automatically generatedCompromised PKC0. When a Controller needs to execute the rotation of their Controller Key associated with their DID document. (A specific example could be to change the Root key for all their DID Issuance). For this scenario the controller will have the same flow presented inside the “loop” of the above diagram, except, the Controller can perform the steps to create the Rotation Tx and publish it on chain by itself. They would still require the Subject to sign the updated DID Document. So will create a new linked Publish Tx to the Rotation Tx. *See Figure 10*

Figure 10

#### 3.4.4 Revoke

In this section we explain how to revoke a DID. This method uses the spent status of the latest UTXO in the DID Document to indicate the active status of a DID. When the last transaction in the chain is spent, the DID is revoked. See section 3.4.2, DID and DID Document Status. A DID can be revoked by either the DID Subject or the DID Controller. Here are three ways the DID Subject can revoke its DID:

* **Subject is still in control of the private key and decides to revoke its DID**, through the DID controller by calling the controller's DID revocation service. Bellow we provide an implementation example. *See Figure 12*
* **Subject is not in control of the private key** (A specific example might be when the subject lost his/her device) and decides to revoke their DID.
* **Subject asks DID Controller to revoke on its behalf**. Note that to secure subject request we advise Controllers to apply a mandatory, strong authentication of DID Subject before executing any changes. Bellow we provide an implementation example. This will be signified by the transaction type indicator, as described in section 3.4.1. *See figure 13*

In all cases the revocation is initiated by the DID Subject.

A screenshot of a diagram

Description automatically generated

Figure 12

A screenshot of a computer

Description automatically generated

Figure 13

## 4. Governance model using DID BSV method.

The UTXO-Based DID Method supports a governance service where the DID Subject accepts a DID from a trusted Controller as valid. The Verifier will only accept as valid a DID Subject whose DID Document is co-signed by that specific controller. As mentioned before, the DID controller can be run by any entity. The BSV DID method does not require a centralized controller.   
Below there’s a description of a specific example for governance:

* Imagine a scenario where the DID Controller is run by the Spanish Directorate-General for Traffic (DGT).
* Every DID issued by the DGT is always co-signed by this controller. The verifier that checks the validity of the DGT operator's DID ensures it is always co-signed by the DGT controller.
* The DID controller establishes an ecosystem where value is derived from attestation to the keys of a DID subject and the signature of the controller. Initially, the controller will issue a DID for itself following the process described in the UTXO DID method. See section 3.4.1.
* In this case, the DID document will be self-attested. The controller will issue a DID to an alternate key-pair, which will then be used as the master keypair for attestation of DIDs to its audience.
* The trust framework supports a schema of key hierarchies and the issuance of the DID Controller. Once the Controller has been assigned a key, it becomes the master of that key for issuing all DIDs to its DID subjects.

**DID <PKCD> = Master key and <PKC0>= Controller Key to sign Subject DIDs.** When the controller signs a new DID Document, the controller will publish the DID public key to enable the authentication of the Controller’s DID.

For governance verification the verifier will use the DID to fetch the transaction on the ledger. Once they find the transaction the verifier will review the UTXO status and the DID Document. For governance check the verifier has to check the DID issued by for the Subject has been co-signed by one of the published Controller keys.

## 5.Low latency DID resolution

As previously detailed (See Section 3), a DID and its associated DID Document are constructed through two linkedtransactions connected by the spending of a unique UTXO. For the DID to be resolved, it is necessary for both transactions to be mined within a block. The default outcome requires DID transactions to be mined in at least one block to be marked as resolved. Given the inherent nature of blockchain technology, the standard time for mining a block in Bitcoin SV, Bitcoin Core or Bitcoin Cash is approximately 10 minutes.

Each blockchain transaction refers to a previous blockchain transaction. Given a TxID the ledger allows an observer trace back to the previous transaction. The DID method requires every DID status update to be linked to the initial transaction TXID0, and the DID controller requires verification of the UTXO status corresponding to the most recent transaction on the chain. Therefore, we are required to trace back though every transaction in the chain until we get to TXID0*.* To make this more efficient, our implementation uses an indexing service to enable forward Tx retrieval and DID status check.

##### Timing and Latency in DID Resolution Processes

Currently, the DID resolver only searches for transactions in mined blocks. Given that the average time for a block to be mined is approximately 10 minutes, this forms the basis for the resolution time it takes for newly minted DIDs. Since broadcasted transactions are not subject to reorganization or modification in the mempool the mempool becomes the first trusted source of transaction status check. In addition, for this specific implementation and the blockchain network of used, which supports large blocks, allows most transactions in the mempool to be included in the following block, enhancing reliability in the mempool. The key innovation of this approach is that it accomplishes low resolution latency, but it shifts the responsibility of the stability of the DID Document from the DID Resolver to the requester of the status check.

##### User-Controlled Security through Miner Block Validation

This approach shifts the responsibility for determining DID security and stability from the BSV DID method to the user. By using miner block validation, users can assess the security of a DID based on the number of block confirmations. This method provides low resolve latency while giving users the flexibility to determine stability based on the DID Resolution Metadata. The metadata includes information on the number of confirmations (i.e., the distance from the mempool in blocks) that a resolved DID document has received. The BSV DID method specifications will also offer clear guidelines on how different numbers of confirmations correspond to varying levels of stability and security guarantees:

* BSV DID Method specification
  + **Resolution process:** should specify that mempool will be searched too
  + **DID Resolution metadata:** should document the new field defined in previous section
  + A screenshot of a computer screen

    Description automatically generated**Security section.** should list recommendations on how number of confirmations maps to DID stability and DID Document stability

A screenshot of a phone

Description automatically generated

The DID Spec Registries should define a new field for [DID Resolution Metadata](https://www.w3.org/TR/did-spec-registries/#did-resolution-metadata) where the resolver can signal the number of confirmations for the first DID transaction (Tx0) and the confirmations of the resolved (last by default) DID Document transaction.

"confirmations": {

- "create": 123,    // confirmations of the initial transaction for the resolved DID

- "update": 234     // confirmations of the DID Document transaction for the resolved DID}

For a newly created DID the number of confirmations is highly likely to be the same as both initial transactions are submitted to the bitcoin network together. The update field will signal the confirmations of the resolved DID Document, which by default is the last DID document in the chain, unless resolution of specific DID document version was requested by Version-ID.

The issuers and verifiers can configure number of valid blocks needed to accept a DID as resolved. It could be 0 confirmation requirement to <n> confirmations.

## 6. Data register analysis

This section outlines the technical reasons for choosing Bitcoin SV, including relevant cost and security considerations.

#### *6.1 Distributed Ledger Technology*

Between 2017 and 2018, three distinct blockchain protocols emerged from Bitcoin's original protocol, sharing a history that dates to the genesis block. These are BTC, BCH, and BSV (Bitcoin SV). The BSV blockchain ledger is a distributed platform that provides scalability, transparency, immutability, and efficient data consumption. Data can be published on the blockchain and retrieved within blockchain transactions.

Bitcoin SV typically produces blocks larger than 1GB, with the maximum recorded block size being 4 GB [3], and it supports a high transaction throughput of up to 5,100 blockchain transactions per second (tps) [4]. It accommodates large transaction sizes, with a record of a 42MB transaction [3] and has a default maximum script size of 500kB [3]. It also includes a complete set of opcodes and supports big integer arithmetic in-script [3]. Previously, we presented a solution based on the Bitcoin SV protocol for blockchain transactions, which can be easily adapted to any UTXO-based blockchain. illustrates a simplified schematic of a Bitcoin transaction. This transaction fee is the lowest among Proof of Work (PoW) blockchains [3].

#### *6.2 Why not others? Ledger Comparison Analysis*

Given the project's requirements, a Proof of Work (PoW), public, and permissionless blockchain was deemed necessary. PoW blockchains record cumulative PoW directly on the blockchain, making the data structure statistically resistant to attacks or modifications. BTC was excluded due to high transaction fees and a limited block size. Apart from Bitcoin SV, the remaining options were:

* **BCH**: A UTXO- PoW ledger with a maximum block size of 32MB. Although BCH has a higher transactions per second (TPS) rate than BTC, its transaction processing remains limited and while transaction fees are lower in average than BTC, they are still relatively high for the needs of this project.
* **Ethereum (ETH)**: Previously a PoW ledger, now using Proof of Stake (PoS) as its consensus mechanism. All statements and data refer to ETH before this change. ETH is an account-based PoW ledger with an integrated smart contract layer that maintains a global state, posing scalability challenges [3]. The new adopted consensus mechanism allows ETH to process block faster than BTC and BCH and BSV. The transaction structure of an Ethereum will request, most likely to utilise a token protocol to store and monitor the DID Document. ERC-20, their most adopted token protocol overage transaction fee remains considerable large for the purpose of this project and as the complexity of the smart contract increases, so does the gas fee. ETH has the highest average transaction fees among blockchains.
* **Private ledgers**, such as Hyperledger, were not considered since solutions built on these ledgers incur higher costs for implementation, maintenance, transaction fees and conventionally single authorities running the infrastructure.

Our solution requires high-volume transaction processing at low cost, as well as maintaining data integrity over time and across parties involved. Immutability and alteration records should be enforced by an independent process: the consensus mechanism. Additionally, it does not require the execution of any programmatic function at the miner level, avoiding execution costs in transaction fees. Lastly in BSV there’s the 'first-seen rule' which means that miners will not accept a double-spend before block publication even for a higher fee if a miner receives a transaction, they verify it is correct, then accept it as valid. This means they will not accept a double-spend of the same input. Then add the transaction in the mempool. It may appear in the next block, or the block after. For these reasons, we decided to implement the solution using Bitcoin SV, benefiting from its scalability, low cost, and data integrity provided by PoW.

## 7. Privacy Considerations

One of the main features of this DID method is to enhance the revocation process to be immediate for the verifier, the issuer and the subject. The verification process is independent of the issuer and does not necessarily require a trusted service provider. This means that no single entity can track the holder’s usage of their credentials. The request of the status check on a specific credential is intentionally made without providing context to the blockchain network (blockchain nodes) and completely independently from the Issuer. Even when a single blockchain node is queried, the node would not be able to link the query to a credential or to a credential holder. This enhanced in privacy would not be possible with traditional approaches.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## UTXO DID Method Normative Reference

### A.1 DID Syntax

|  |  |
| --- | --- |
| DID method name: | **bsv** |
| DID method specific identifier | 64 characters long hexadecimal presentation of BSV Blockchain transaction ID |

The hexadecimal presentation MUST use lowercase characters for hexadecimal digits *a–f.*

Example: did:bsv:8280eadd84515d071e942f1548af7891e0182bf83e7b5848a02b0a58543fbf5f

### A.2 Verifiable Data Registry

UTXO DID method SHOULD use BSV Blockchain as a public ledger to store the DID related data and statuses. Each DID is recorded as a sequence of chained transactions of the following kinds:

* *DID issuance transaction*, which is the first transaction in the DID chain
* *DID document transaction*, a transaction which OP\_RETURN data contains a version of the DID document
* *DID revocation transaction*, which MUST be the last transaction in the DID chain and which presence invalidates DID

Owing to the blockchain design, each of the DID document transactions MUST be preceded by either DID issuance transaction or a special *DID funding transaction* which provide additional funds for mining fees. DID funding transactions do not change the state of DID.

Below text uses the following symbols that refer to SECP256K1 private and public keys:

|  |  |
| --- | --- |
| C0 , PKC0 | DID controller’s private and public key, respectively. |
| S0, PKS0 | DID subject’s private and public keys, respectively. |
| SigC0, SigS0 | Signatures made by using DID controller’s or DID subject’s private keys, respectively. |
|  |  |

It also uses term *identityCode*. This is a code that identifies the DID controller that created by transactions. This code MUST be present and it SHOULD be unique.

#### A.2.1 DID Issuance Transaction

The issuance transaction has one input and one output. The input spends the funding UTXO provided out-of-band by the issuance transaction creator. The UTXO spent by this transaction MUST contain enough funds to cover mining fees for this and two next transactions.

The OTXO of DID issuance transaction MUST provide funds to cover mining fees for two next transactions.

The locking script of DID issuance transaction MUST require signatures by both DID controller and DID subject to spend it.

The first segment of the OP\_RETURN data payload MUST be set to the string literal BSVDID.

The second segment of the OP\_RETURN payload MUST be set to *identityCode*.

The third segment of the OP\_RETURN data payload MUST be set to string literal 1 (numerical digit 1).

Implementations MAY add more segments for their own use.

|  |  |  |
| --- | --- | --- |
| **Tx0** | | |
| Version: 1 | | |
| In-count: 1 | | Out-count: 1 |
| **Input list** | | **Output list** |
| 0 | Outpoint: *funding UTXO* | Value: *mining fee for two transactions* |
| Unlocking script: *depends on funding UTXO* | Locking script:  2 *<PKC0> <PKS0>* 2 OP\_CHECKMULTISIG  OP\_RETURN  BSVDID  <*identityCode*>  1 |
| Sequence number: 0xFFFFFFFF |

#### A.2.2 DID Document Transaction

The DID document transaction is funded either by the DID issuance transaction or DID funding transaction. It carries DID document in the third segment of OP\_RETURN payload. The transaction has one input which MUST spend the UTXO provided by either DID issuance or DID funding transaction.

The OTXO of DID document transaction MUST provide funds to cover mining fee for the next transaction.

The locking script of DID document transaction MUST require signature by either DID controller or DID subject to spend it.

The first segment of the OP\_RETURN data payload MUST be set to the string literal BSVDID.

The second segment of the OP\_RETURN payload MUST be set to *identityCode*.

The third segment of the OP\_RETURN data payload MUST be set to the JSON string representing DID document.

Implementations MAY add more segments for their own use.

|  |  |  |
| --- | --- | --- |
| **TxD*i*** | | |
| Version: 1 | | |
| In-count: 1 | | Out-count: 1 |
| **Input list** | | **Output list** |
| 0 | Outpoint: Tx0//0 or TxF*i*//0 | Value: *mining fee for the next transaction* |
| Unlocking script: SigC0 **and** SigS0 | Locking script:  1 *<PKC0> <PKS0>* 2 OP\_CHECKMULTISIG  OP\_RETURN  BSVDID  <*identityCode*>  *<DID document>* |
| Sequence number: 0xFFFFFFFF |

The transaction ID (TxID) of the DID document transaction is the *versionId* of the DID document and resolvable as such.

The timestamp of the block that contains the DID document transaction is the *versionTime* of the DID document and resolvable as such.

#### A.2.3 DID Revocation Transaction

The DID revocation is funded by the preceding DID document transaction. The transaction has one input which MUST spend the UTXO of the preceding DID document transaction. The transaction MUST one output with value 0, which makes it unspendable. The DID revocation transaction, when present, is thus the last transaction in the DID chain and invalidates the DID.

The OTXO of DID revocation transaction MUST have value 0 and SHOULD have no other locking script but OP\_RETURN.

The first segment of the OP\_RETURN data payload must be set to the string literal BSVDID.

The second segment of the OP\_RETURN payload MUST be set to *identityCode*.

The third segment of the OP\_RETURN data payload MUST be set to string literal 3 (numerical digit 3).

Implementations MAY add more segments for their own use.

|  |  |  |
| --- | --- | --- |
| **TxR** | | |
| Version: 1 | | |
| In-count: 1 | | Out-count: 1 |
| **Input list** | | **Output list** |
| 0 | Outpoint: TxD*i*//0 | Value: 0 |
| Unlocking script: SigC0 **or** SigS0 | Locking script:  OP\_RETURN  BSVDID  <*identityCode*>  3 |
| Sequence number: 0xFFFFFFFF |

#### A.2.4 DID Funding Transaction

The DID funding transaction does not affect the DID state and is used only to bring in additional funds needed to extend the DID transaction chain. IT has two inputs and one output. The first input MUST spend the out-of-band UTXO provided by the creator of this transaction. This UTXO MUST provide enough funds to cover mining fee for two transactions. The second input MUST spend the UTXO of the preceding DID document transaction, which provides funds to cover mining fee for one transaction.

The OTXO of DID funding transaction MUST provide funds to cover mining fee for two next transactions.

The locking script of DID funding transaction MUST require signature by both DID controller and DID subject to spend it.

The first segment of the OP\_RETURN data payload MUST be set to the string literal BSVDID.

The second segment of the OP\_RETURN payload MUST be set to *identityCode*.

The third segment of the OP\_RETURN data payload MUST be set to string literal 2 (numerical digit 2).

Implementations MAY add more segments for their own use.

|  |  |  |
| --- | --- | --- |
| **TxF*i*** | | |
| Version: 1 | | |
| In-count: 2 | | Out-count: 1 |
| **Input list** | | **Output list** |
| 0 | Outpoint: *funding UTXO* | Value: *mining fee for two transactions* |
| Unlock script: *depends on funding UTXO* | Locking script:  2 *<PKC0> <PKS0>* 2 OP\_CHECKMULTISIG  OP\_RETURN  BSVDID  <*identityCode*>  2 |
| Sequence number: 0xFFFFFFFF |
| 1 | Outpoint: TxD*i*//0 |
| Unlocking script: SigC0 **or** SigS0 |
| Sequence number: 0xFFFFFFFF |

#### A.2.5 Example of DID Transaction Chain

THIS SECTION IS NOT NORMATIVE

The below example shows the DID transaction chain for DID that had one change in DID document and was finally revoked.

**TxD2**

Input 0

UTX 0

Value: $

**TxF1**

Input 1

Input 0

UTX 0

Value: $

**TxD1**

**Tx0**

UTX 0

Value: $

Input 0

Input 0

UTX 0

Value: $

**TxR**

Input 0

UTX 0

Value: 0

### A.3 DID Operations

The authorisations to perform the operations rely on the properties of the blockchain that will reject the transactions which unlocking script will fail to unlock the addressed UTXOs. Both DID controller and DID subject must possess their own SECP256K1 keys which they MUST use to sign the DID transactions that are to be added to the DID’s chain of transactions.

When DID subject and DID controller are implemented as separate entities, they each use their own DID, which when resolved will yield their respective public keys. The DID subject’s DID document list the DID of its controller:

|  |
| --- |
| {  "@context": "https://www.w3.org/ns/did/v1",  "id": "did:bsv:1909a27e2f9c01dbc99f6391f77cd96032e00b92ff601584b7e227009055e6da",  "controller": "did:bsv:a2fc941316adc10d680c98a06bf68da88ec513c658a9dc5ef837058064b588f9",  "verificationMethod": [  {  "id": "did:bsv:1909a27e2f9c01dbc99f6391f77cd96032e00b92ff601584b7e227009055e6da#subject-key",  "type": "JsonWebKey2020",  "controller": "did:bsv:1909a27e2f9c01dbc99f6391f77cd96032e00b92ff601584b7e227009055e6da",  "publicKeyJwk": {  "kty": "EC",  "x": "l5sYB-y78yE-n8Fc1d9pZeVFYhGD8qWoFRk-abyr0p4",  "y": "XVcP352DcjrO374NW-4iNfUUWLLu\_ZvcNhZtWPFURAg",  "crv": "secp256k1"  } } ],  "authentication": [  "did:bsv:1909a27e2f9c01dbc99f6391f77cd96032e00b92ff601584b7e227009055e6da#subject-key"  ]  } |

In SSI use, the DID document lists both public keys since the DID controller’s DID does not exist:

|  |
| --- |
| {  "@context": "https://www.w3.org/ns/did/v1",  "id": "did:bsv:a2fc941316adc10d680c98a06bf68da88ec513c658a9dc5ef837058064b588f9",  "verificationMethod": [  {  "id": "did:bsv:a2fc941316adc10d680c98a06bf68da88ec513c658a9dc5ef837058064b588f9#subject-key",  "type": "JsonWebKey2020",  "controller": "did:bsv:a2fc941316adc10d680c98a06bf68da88ec513c658a9dc5ef837058064b588f9",  "publicKeyJwk": {  "kty": "EC",  "x": "oA39rl7p8GjPi4oVJRhesnhm9M4RKl0\_dWs6EjDSm84",  "y": "l9j72B38c3VHlwk9-YtWyLovLFGU\_Bkx1qa6DRLf5UQ",  "crv": "secp256k1"  }  }  ],  "service": [  {  "id": "did:bsv:a2fc941316adc10d680c98a06bf68da88ec513c658a9dc5ef837058064b588f9#controller-website",  "type": "LinkedDomains",  "serviceEndpoint": "*<service-endpoint-url>*”  }  ],  "authentication": [  {  "id": "did:bsv:a2fc941316adc10d680c98a06bf68da88ec513c658a9dc5ef837058064b588f9#auth",  "type": "JsonWebKey2020",  "controller": "did:bsv:a2fc941316adc10d680c98a06bf68da88ec513c658a9dc5ef837058064b588f9",  "publicKeyJwk": {  "kty": "EC",  "x": "-femws1TMAIQZLWNuhglkF9N6RrJAIJA7Yk64CavSEA",  "y": "BvoLuvdFvLv1p3j1myLZrhCBmYvoBHtp15ol\_BB81Yk",  "crv": "secp256k1"  }  }  ]  } |

#### A.3.1 DID Creation

To create DID, DID controller MUST publish two transactions to the blockchain:

* DID issuance transaction (see A.2.1 DID Issuance Transaction), and
* DID Document transaction (see A.2.2 DID Document Transaction) containing the first version of the DID document)

It is the DID controller’s responsibility to provide funding to cover mining fees for those transactions.

When the DID subject is acting as own controller (SSI) it MUST own and use, C0/PKC0 representing DID controller and S0/PKS0 representing DID subject.

#### A.3.2 DID Document Updates

To create a new version of DID document, DID controller must publish two transactions to the blockchain:

* DID funding transaction (see A.2.4 DID Funding Transaction) which provides funds to cover mining fees of this and a new DID document transaction.
* DID Document transaction (see A.2.2 DID Document Transaction) which publishes new version of DID document

It is the DID controller’s responsibility to provide funding for DID funding transaction.

While the DID funding transaction can be signed by either DID subject or DID controller, the DID document transaction itself must be signed by both. This ensures that both DID subject and DID controller agree on the new DID document contents.

#### A.3.3 DID Deactivation

To deactivate, or revoke a DID, either DID subject or DID controller MUST publish DID revocation transaction (see A.2.3 DID Revocation Transaction). The funds for this transaction are already available in the UTXO of the last transaction of the DID’s transaction chain.

#### A.3.4 DID Resolution

To resolve DID to the latest version of the DID document, DID resolver MUST:

1. Fetch the blockchain transaction which transaction ID is the method specific identifier of DID, decoded from hexadecimal presentation to binary number and make it current.
2. Fetch the transaction that spent UTXO 0 of the current transaction
3. Check the transaction kind
   1. If DID document transaction (see A.2.2 DID Document Transaction), check if current transaction’s UTXO 0 is unspent
      1. If yes, return the DID document as the result of the resolution
      2. If no, make this transaction current and repeat from step 2
   2. If DID funding transaction (see A.2.4 DID Funding Transaction) make this transaction current and repeat from step 2
   3. If DID revocation transaction, return the last seen DID document as the result of the resolution and set deactivated property to true in DID document metadata and terminate

If versionId query is present, the algorithm must modify steps 3.1 to read:

3.1 If DID document transaction check if its transaction ID matches the value of versionId query

3.1.1 If yes, return the DID document as the result of the resolution

3.1.2 If no, check if the UTXO of this transaction is spent

3.1.2.1 If no, set error property of DID resolution metadata to notFound and terminate

3.1.2.1 If yes, mark transaction as current and repeat the step 2

And step 3.3 to read:

3.3 If DID revocation transaction, set error property of DID resolution metadata to notFound and terminate

## 

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |