Work in progress

This is an early draft of a chapter of a book on the foundations of cryptographic authentication being coauthored by Francisco Corella, Sukhi Chuhan and Veronica Wojnas. Please send comments to the authors.

14. Decentralized identifiers

In this chapter we are going to prepare for the topic of the next chapter, *verifiable credentials*, by taking a close look at the identifiers that are used to refer to the subjects of those credentials, *decentralized identifiers (DIDs)*.

The W3C "Decentralized Identifiers (DIDs) v1.0" specification[1] defines a decentralized identifier as "a globally unique persistent identifier that does not require a centralized registration authority and is often generated and/or registered cryptographically". DIDs are complex because the criteria of this definition are tough to meet.

For example, we saw in Chapter 5 that the most common traditional credential, a public key certificate, binds attributes of the subject to a public key. Is that public key a decentralized identifier?

It almost meets the above definition, but not quite. It is a globally unique identifier that refers to the subject of the certificate, and it is generated cryptographically, along with its associated private key, by the subject's own computing equipment, without relying on a registration authority. But it is not a *persistent* identifier. It is generated when the certificate is issued and only used in that certificate. We shall look in Section 14.2.2 at a DID, called did:key, that, in first approximation, is just a public key; except for the fact that it is used in multiple credentials. And we shall see in Chapter 15 how that fact makes a big practical difference.

14.1 Early constructions of decentralized identifiers

14.1.1 Namecoin

The first demonstration of decentralized identifiers may be credited to Namecoin[2], a blockchain derived from and merge-mined with Bitcoin, specifically designed to support the registration of human-meaningful, memorizable decentralized identifiers. Namecoin disproved Zooko's trilemma[3], the conjecture stated by Zooko Wilcox-O'Hearn that decentralized identifiers that could be memorized by humans could not be securely implemented.

The conjectured security challenge was how to prevent two users from registering the same memorizable name, a challenge similar to the double-spend problem: how to prevent the same digital coin from being spent twice[4]. A blockchain or distributed ledger can solve both problems by achieving consensus on which of two name-registration transactions, or two coin-spending transactions, is deemed to have occurred first, the other one being invalid.

Namecoin was used to implement a variety of memorizable namespaces. The first of those was a decentralized DNS for a top-level domain ".bit" not sanctioned by ICANN[5]. Namecoin is an "altcoin"[6], and as such it provided a marketplace for domain names ending in ".bit". That marketplace failed due to widespread squatting of potentially valuable domain names[7], and support for ".bit" domain names was eventually dropped from OpenNIC[8]; but Zooko's conjecture had been disproved.

Namecoin was also used to implement a namespace for login usernames, and an OpenID identity provider that mapped those usernames to profiles stored in the blockchain,[9] anticipating current proposals to use more recent versions of OpenID for the issuance and presentation of verifiable credentials[10].

14.1.2 Decentralized PKI

While Namecoin was a general-purpose facility for mapping human-meaningful names to any kind of values, the seminal DPKI white paper[11] from the collaborative project "Rebooting the Web of Trust"[12] more specifically proposed using a blockchain or other kind of decentralized data store to map decentralized identifiers to public keys, as an alternative to traditional public key infrastructures.

The white paper specifically mentioned email addresses, usernames, and website domain names as decentralized identifiers that would be mapped to public keys. It envisioned mapping each identifier to two kinds of keys: a key pair that would be used for registering the identifier, and "subkeys" that could be used to sign messages by the "principal" represented by the identifier. These keys and subkeys can be viewed as the precursors of the public keys contained in the "document" associated with a W3C DID as described in the next section.

14.2 W3C Decentralized Identifiers

Early DIDs were expected to be memorizable, human-meaningful[13], or human-friendly. That was a compelling reason for registering them in blockchains or distributed ledgers. But that is no longer a requirement: Section 9.10 of the W3C DIDs v1.0 specification[14] explicitly states that "human-friendly identifiers" are out of scope.

However, although there are now DIDs that are not registered on a blockchain or distributed ledger, most of them still are. This may be due to the tradition and know-how inherited from the early days of human-friendly DIDs, but there are other reasons besides avoiding collisions of memorizable names that motivate registration in decentralized data stores.

In any case, understanding DIDs requires understanding how they can be registered on a blockchain, so we shall use the Bitcoin Reference DID[15], did:btcr, as a prototypical example of a W3C DID in the following section.

14.2.1 did:btcr

A W3C DID is a globally unique string that has an associated "DID document". The string is formatted as a Uniform Resource Identifier (URI)[16] comprising three parts, separated by colons: the URI scheme "did", the name of a "DID method", such as "btcr" in our prototypical example, and a method-specific identifier. The DID document is a data structure serializable into a variety of text formats, such as JSON-LD[17] of JSON. The DID method specifies how the DID is created, updated, deleted, and "resolved" to the DID document. Resolving the DID means locating or constructing the DID document given the method-specific identifier.

14.2.1.1 DID document

In general, the document of a DID contains or refers to information related to the subject of the DID, such as public keys and URLs of service endpoints. The public keys are called "verification methods" in the DID specification, and are related to the subject by "verification relationships". An "authentication relationship" relates the subject to a public key called an "authentication method", which can be used to authenticate the subject by proof of possession of an associated private key for purposes such as logging in to a web site. An "assertion relationship" relates the subject to a public key, called an "assertion method" that can be used to verify the signatures on verifiable credentials *issued by the subject*; the subject of a DID might want to issue verifiable credentials to participants in a "web of trust"[18] having their own DIDs, or issue a self-signed verifiable credential binding profile information to the DID.

In did:btcr, the DID subject may have a Bitcoin wallet with private keys for signing blockchain transactions, and the public key associated with one of those private keys may be used as the authentication method and/or the assertion method in the DID document.

A DID document may also have a service endpoint for receiving DIDComm messages as described in Section 14.3, a public key that senders can use to encrypt sent to the DID subject messages, and a public key that recipients can use to verify signatures on messages sent by the subject.

14.2.1.2 DID creation

A Bitcoin user creates a BTCR DID of which it becomes the subject by generating a key pair, encoding the SHA256-RIPEMD160 double hash of the public key into a fresh Bitcoin address, and signing an initial transaction that transfers a nominal amount of bitcoin to the fresh address. After the initial transaction has been confirmed by the blockchain, the method-specific identifier of the new DID is a reference to that transaction, comprising the block height in the chain and the transaction index in the block.

By default, the new DID has a DID document where a Bitcoin public key, the one associated with the private key used to sign the initial transaction, is repurposed as an authentication method and an assertion method. The default DID document is constructed by resolver software, which finds the public key in the scriptSig of the transaction; Bitcoin scripts are explained in Chapter 9.

A DID document other than the default can be used by adding a second output to the transaction with an OP_RETURN opcode whose data field is a URL pointing to the document.

14.2.1.3 DID update

After any time after the DID has been created, the subject may need to update the DID document. Before any updates, the transaction referenced by the method-specific identifier of the DID has an unspent transaction output (UTXO) for the nominal amount of bitcoin and an optional unspendable OP_RETURN output. To update the DID document, the subject signs a second transaction that further transfers the nominal amount of bitcoin from the UTXO of the first transaction to a fresh address. That second transaction has a UTXO for the nominal amount, and an OP_RETURN output whose data field references the updated DID document. Further updates can be made by issuing a chain of transactions, each spending the UTXO of the previous one and having an OP_RETURN output referencing an updated document.

14.2.1.3.1 The "controller" concept

The DID specification defines the DID controller as "the entity (person, organization, or autonomous software) that has the capability—as defined by a DID method—to make changes to a DID document".

In typical use cases relevant to this book, the role of DID controller is performed by the DID subject. However, there are important use cases where the controller is not the subject. For example, DIDs can be used to track the provenance and usage of plastic recyclates[19]. In such a use case the DID controller is an organization, while the subject is a product.

14.2.1.3.2 Key rotation in did:btcr

A reason for updating the DID may be to add a service endpoint or a new verification method to the DID document. Another reason may be to update a verification method by "rotating", i.e. replacing, the public key that participates in the corresponding verification relationship and its associated private key. A benefit of a DID registered in a data store and used in a verifiable credential, when compared with a public key used in a traditional public key certificate, is the ability to rotate the private key used for authenticating the subject without revoking and reissuing the credential.

In general, rotation of a key pair may be needed for three reasons:

1. To limit the number of times that the key pair is used, as a countermeasure against cryptanalysis.
2. To replace the cryptosystem that the key pair pertains to if it has been deprecated.

3. To replace the private key if it has been compromised.

In did:btcr, however, it may not be safe to use key rotation to recover from compromise, as compromise of a private key associated with a verification method may be due to compromise of the Bitcoin wallet that would be used to issue the Bitcoin transaction that would update the DID document. When key compromise is detected, the DID should be abandoned and verifiable credentials that assert claims about the DID should be revoked.

14.2.1.4 DID deletion

An OP_RETURN output is optional in the transaction used to create the DID, but is not optional in the chain of subsequent transactions used to update it. By convention, a DID is deemed to be "deleted" if that chain ends with a transaction that does not have an OP_RETURN output. A DID that has never been updated is deleted by issuing a transaction as if to update it but omitting the OP_RETURN output.

14.2.1.5 DID resolution

The DID specification defines a "DID resolver" as "a system component that takes a DID as input and produces a conforming DID document as output" and refers to the process used by a DID resolver as "DID resolution".

Section 4.2 of the did:btcr specification[15] sketches a process like the following one to resolve a BTCR DID:

1. Look up the confirmed Bitcoin transaction referenced by the method-specific identifier of the DID.
2. If the transaction has an unspent output, no updates have been performed. In that case:
    a. If the transaction does not have an OP_RETURN output, construct the default DID, using the public key in the scriptSig of the transaction as both the authentication method and the assertion method.
    b. Otherwise return the DID document referenced by the data field of the OP_RETURN opcode.
3. Otherwise the transaction referenced by the DID is the first of a chain where each transaction has spent a UTXO of the previous one. Follow the chain until a transaction is found with an unspent output.
    a. If that transaction does not have an OP_RETURN output, the DID has been deleted.
    b. Otherwise return the DID document referenced by the data field of the OP_RETURN opcode.

14.2.2 did:key

The did:key Method v0.7 specification[20] provides the simplest example of a DID that is not registered on a blockchain, distributed ledger, or other kind of decentralized registry. In first approximation, a did:key DID is just a public key, which plays the same role in a verifiable credential as a public key in a traditional public key certificate. A did:key DID cannot be updated, and thus does not provide the benefit of key rotation. However, it is a persistent identifier usable in multiple verifiable credentials, and we shall see in Chapter 15 how that provides important benefits, akin to those provided by the combination of a traditional public key certificate and multiple attribute certificates[21].

Like all W3C DIDs, a did:key DID is a URI with three parts separated by colons: the "did" scheme; the name of the DID method, in this case "key"; and the method-specific identifier, which in this case is an encoding of a public key. The DID document is derived from the public key encoded into the DID without reliance on a registry; but the algorithm used by the resolver takes options as an input, which must be agreed upon for interoperability.

We saw in Section 4.2.1.2 how the default document of a BTCR DID is constructed by repurposing a Bitcoin public key as an authentication method and an assertion method. In did:key, the DID document is similarly constructed by repurposing the public key encoded in the method-specific identifier as several verification methods. The public key must be usable for digital signature, and it is repurposed as an authentication method and an assertation method, like the Bitcoin public key in did:btcr. As shown in Example 2 of the specification, it is also repurposed as "capability invocation" and "capability delegation" verification methods, usable to authorize access to a protected resource by the subject or a delegate.

But in the particular case where the public key pertains to the Ed25519 cryptosystem, it is also used to derive a corresponding X25519 public key used as a "keyAgreement" or "encryption" method. (The terms "keyAgreement method" and "encryption method" are used synonymously in the specification because key agreement can be used to derive or wrap a symmetric encryption key.)

This deserves an explanation. (See also the glossary of Section 4.2.2.1.)

X25519 is an ECDH key agreement cryptosystem based on Curve25519, a Montgomery curve[22] over the field defined by the prime number 2^{255}-19. Ed25519 is a "high-speed" elliptic curve signature cryptosystem[23] that uses a "twisted Edwards curve"; it is a special case of EdDSA, which is akin to but different from ECDSA. The paper by Bernstein et al. that introduced twisted Edwards curves[24] shows that every such curve is birationally equivalent[25] to a corresponding Montgomery curve. The curve used in Ed25519, called edwards25519, is the one that corresponds to the Curve25519 of X25519. Formulas defining the birational map between ed25519 and Curve25519 are provided in Section 4.1 of RFC 7748[26].

The public key of Ed25519 encoded into the DID of Example 2 is the x coordinate of a point of edwards25519, and the corresponding X25519 public key used as a keyAgreement method is the u coordinate of the birational image of that point in Curve25519. It is generally deemed unsafe to use the same cryptographic material for two different purposes, but Section 3.1.6 or the specification cites a paper[27] arguing that it is safe in this particular case to derive a key agreement public key from a signature public key.

Sections 3.1.1-7 of the specification refer to the intricate details of decoding the DID of Example 1 and encoding the DID document of Example 2. It may be useful to know that Section 3.1.6 mistakenly refers to Section 2.4.2 of [OSCORE]; it should be 2.5.2.

14.2.2.1 A glossary of "25519" terminology

- $2^{255} – 19$: a prime number.
- Curve25519: a Montgomery curve over the field defined by $2^{255} – 19$.
- edwards25519: a twisted Edwards curve over the field defined by $2^{255} – 19$.
- Ed25519: an elliptic curve signature cryptosystem that uses the curve edwards25519.
- EdDSA: a generalization of Ed25519 to other choices of curves, akin to but different from ECDSA.
- X25519: the ECDH cryptosystem that uses Curve25519.

14.2.3 KERI

KERI is an identity system where public keys are encoded into decentralized identifiers as in did:key, but key rotation is supported. It is still a work in progress, which has been evolving over the last 5+ years and is still undergoing major changes, with the introduction of many new concepts and acronyms[28]. KERI is an acronym for "Key Event Receipt Infrastructure", which is the title of a seminal whitepaper that evolved from version 1 dated 3 July 2019 to version 15 dated 11 October 2021[29]. A specification of a did:keri Method v0.1 dated 10 November 2021[30] was published on the web site of the Decentralized Identity Foundation. A subsequent specification of a did:keri DID Method was published on 4 May 2023 as an Internet Draft[31], with references to KERI Improvement Documents (KIDs) published on a GitHub repository that was archived on April 23, 2023[32]. A successor to the seminal KERI white paper has been published as an Internet Draft of same title[33]**Error! Bookmark not defined.**, along with Internet Drafts that define the concepts of a Self-Addressing IDentifier (SAID)[34] and Authentic Chained Data Containers (ACDC)[35].

14.2.3.1 Key rotation and pre-rotation in KERI

Key rotation is implemented in KERI using a "key event log (KEL)" where rotations of one or more key pairs are recorded as key events. Section 7.41 of the original KERI white paper[29] suggests that the KEL could be stored in the Interplanetary File System (IPFS)[36], where it could be accessed by a "KERI resolver". Section 4.2 of the KERI DID specification of 4 May 2023[31] states that the method for discovering the KEL is out of scope and suggests several possible implementations.

In the basic case where a single key pair it being rotated in each key event, each rotation establishes the "current key pair", generates a "next key pair" that will become the current key pair at the next rotation, and uses the current private key to sign a key event comprising the current public key and a hash of the next public key. Generating ahead the next key pair is called "pre-rotating". The purpose of pre-rotating is to ensure that the next private key has never been used when it is installed as the current private key, and therefore cannot have been compromised by an attack of a kind that can only take place when the private key is being used.

The two versions of the KERI white paper argue that this makes it safe to use key rotation to recover from private key compromise. However, while there are kinds of attacks, such as side-channel attacks, that can only take place as the private key is being used, there are other kinds of attacks, such as key exfiltration from a compromised cryptographic module, that can take place whether the private key is being used or not. If such attacks are possible, compromise of the current private key should be taken as an indication that the next private key has been compromised as well. The safest course of action in that case would be to abandon the identifier and revoke the verifiable credentials that use it.

14.2.4 did:peer

The Peer DID Method Specification[37] defines several kinds of DIDs intended to be used by a limited number of "peers" that participate in a digital relationship. A peer DID may be a "pairwise DID", which is "intended to be known by its subject and exactly one other party"; or an "N-wise DID", which is "intended to be known by exactly *N* enumerated parties including its subject".

Peer DIDs are constructed, or "generated", from their DID documents. The specification describes five "generation methods", numbered 0 to 4. Like other DIDs, a peer DID is composed of a prefix, in this case "did:peer:", followed by a method-specific identifier, which in this case is itself composed of the generation number, in the range 0 - 4, followed by a "generation-specific identifier"[a].

In generation methods 1 and 3, the generation-specific identifier is a cryptographic hash of the DID document. Since a cryptographic hash function is a one-way function, the generation process is not reversible. Hence it is not possible to resolve a DID with generation number 1 or 3 in the normal way, by deriving the DID document from the DID. A peer may be able to resolve such a DID by remembering the DID document, if it was stored during a prior interaction with the peer that owns the DID. Storage of DID documents of other peers is facilitated by the fact that there is a limited number of peers; it would be impractical for "anywise DIDs", which are "intended for use with an unknowable number of parties".

In generation methods 0, 2 and 4, on the other hand, the generation process is reversible, and DIDs can be resolved in the normal way by deriving the DID document from the DID. Section 3.4.5 says that a DID with generation number 4 is "statically resolvable". The same could be said of DIDs with generation number 0 or 2, while peer DIDs with generation numbers 1 or 3 would only be "dynamically resolvable"[a].

14.2.4.1 Generation methods

[a] This terminology is not used in the specification.

A peer DID constructed with *generation method 0* is essentially a did:key DID. It is obtained from a did:key DID by replacing the prefix "did:key:" with "did:peer:0". It is generated from the DID document by encoding the authentication public key into the generation-specific identifier. It is statically resolvable.

Generation method 1 can be applied to any DID document, resulting in a peer DID whose generation-specific identifier is a cryptographic hash of the document. A peer DID constructed with generation method 1 is not statically resolvable. It can be dynamically resolvable if peers share their DID documents, using, for example, the DID Exchange Protocol 1.0 of Aries RFC 0023[38].

A peer DID constructed with *generation method 2* has a very long generation-specific identifier that encodes the concatenation of all the public keys used as verification methods in the document, and the URLs of all the service endpoints. This amounts to encoding the entire DID document into the DID. Such a DID is statically resolvable.

Generation method 3 is a two-step process. First, the concatenation of all the public keys and service endpoints is assembled as in generation method 2. Then a cryptographic hash of that concatenation is computed and used as the generation-specific identifier. This amounts to encoding a "short form" of the DID document into the generation-specific document. A peer DID constructed with generation method 3 is not statically resolvable. Like a method-1 DID, it can be dynamically resolved if peers share their DID documents. Alternatively, a peer who receives a method-3 DID, e.g. in the "from:" attribute of a DIDComm message, may be able to dynamically resolve it if it has previously received a method-1 DID from the same source, and retained the concatenation of the keys and endpoints found in that DID. The specification refers to that dynamic resolution alternative when it states that "Method 3 peer dids can only be used after a peer did method 2 has been exchanged with the other party and thus can map the shortened did to the longform one", where "did" should have been "did document".

Generation method 4 constructs the concatenation of the keys and endpoints in the DID document as in generation method 2 and its cryptographic hash as in method 3, then includes both in the generation-specific identifier. This amounts to including both a short form and a long form of the DID document in the DID. The specification motivates this redundant inclusion by stating in Section 3.4.5 that "The combined use of short and long forms allows for fully peer shared DID Documents, with efficient use of the short form after initial exchange".

4.3 DIDComm

The DIDComm Messaging specification[39] provides a mechanism for subjects of decentralized identifiers to send optionally signed and/or encrypted messages to each other. Examples of such messages can be found in Appendix C (Section 12.3) of the specification.

4.3.1 Message encoding and transport

As specified in Section 3 of the specification, a plaintext message is encoded as a JSON Web Message (JWM)[40]. As shown in the example provided in Appendix C, the payload is placed in the body of the JWM, structured as a collection of name-value pairs. The source and destination(s) are DIDs, specified by "from" and "to" attributes.

DIDComm is transport-agnostic, and Section 1.2 states that it is "usable over HTTPS 1.x and 2.0, WebSockets, BlueTooth, chat, push notifications, libp2p, AMQP, SMTP, NFC, sneakernet, snail mail". Section 8.5.1 cites HTTPS transports as an effective way to send a message to another online agent, and specifies the following requirements for doing so, among others:

- Messages MUST be transported via HTTPS POST.
- POST requests are used only for one-way transmission from sender to receiver; responses don't flow back in the web server's HTTP response.
- Using HTTPS with TLS 1.2 or greater with a cipher suite providing Perfect Forward Secrecy (PFS) allows a transmission to benefit from PFS that's already available at the transport level.

To send a message over TLS, the sender's agent would look for a service of type "DIDCommMessaging" in the DID Document of the recipient, and make a connection to the https URL found in the "uri" property of the "serviceEndpoint" of the service, as shown in the examples of Section 9.4.9 of the specification.

It should be noted that, when a DID has a TLS service endpoint, it cannot be said, strictly speaking, to be decentralized, since HTTPS relies on ICANN to map the domain name of the TLS URL to an IP address, and on a chain of certificate authorities to verify the TLS certificate.

4.3.2 Message signing and encryption

A signed message can be constructed by embedding a plaintext message as the payload of a JWS (JSON Web Signature[41]) message. The message must be signed with a private key (not a symmetric key – signature with a MAC is not supported) and the associated public key must be included as an authentication method in the sender's DID document. The sender's DID must be found in the "from" attribute of the plaintext message, and a URI referencing the authentication method must be included as the value of a "kid" attribute in the JWS message, as stated in Section 3.2 of the specification: "The from attribute in the plaintext message MUST match the signer's kid in a signed message".

Appendix C.2 of the specification shows three examples of signed messages with kid attributes that reference three authentication methods found in a sender DID document shown in Appendix B.1.

Similarly, a plaintext message can be encrypted by embedding it as the payload of a JWE (JSON Web Encryption[42]) message. Two encryption methods are available: "authcrypt", which provides repudiable sender authentication in addition to encryption, and "anoncrypt", which does not provide sender authentication but can be combined with message signing for non-repudiable sender authentication.

Both encryption methods use a static ECDH public key of the recipient, which must be included as a keyAgreement method in the recipient's DID document; the recipient's DID must be found in the "to" attribute of the plaintext message, and a URI referencing the keyAgreement method must be included as the value of a "kid" attribute in the JWE message. Authcrypt further uses a static ECDH public key of the sender, which must be included in the sender's DID document; the sender's DID must be found in the "from" attribute of the plaintext message, and a URI referencing the keyAgreement method must be included as the value of an "skid" (sender key ID) attribute in the JWE message. These "message layer addressing consistency" requirements are also included in Section 3.2 of the specification.

Anoncrypt uses the ECDH-ES algorithm specified in Section 4.6 of the JSON Web Algorithms (JWA) specification[43]. An ephemeral ECDH key pair is generated by the sender, an ephemeral-static key agreement is performed with the recipient, and the plaintext is encoded under a content encryption key that may be derived from the ECDH shared secret or generated at random and encrypted under a symmetric key derived from the shared secret.

Authcrypt uses the ECDH-1PU algorithm of draft-madden-jose-ecdh-1pu-04[44], which uses the "(Cofactor) One-Pass Unified Model, C(1e, 2s, ECC CDH)" scheme of NIST SP 800-56A[45] for key agreement, and is otherwise the same as ECDH-ES. The "1e, 2s" code included in NIST's designation of the key agreement algorithm refers to the fact that it uses one ephemeral and two static key pairs. As used by DIDComm, the sender uses the static key pair referenced by the skid attribute in the JWE to perform a static-static key agreement with the recipient and obtain a first shared secret, and an ephemeral key pair to perform an ephemeral-static key agreement and obtain a second shared secret. Then it uses the concatenation of the two shared secrets to derive or wrap the content encryption key.

---

[1] Decentralized Identifiers (DIDs) v1.0. W3C Recommendation 19 July 2022. https://www.w3.org/TR/did-core/

[2] Namecoin. https://www.namecoin.org/.

[3] Zooko Wilcox-O'Hearn. Names. Secure, Distributed, Human-Readable. Choose Two. https://www.cs.princeton.edu/courses/archive/spr17/cos518/papers/zooko-triangle.pdf

[4] Aaron Swartz. Squaring the Triangle: Secure, Decentralized, Human-Readable Names (Aaron Swartz's Raw Thought). http://www.aaronsw.com/weblog/squarezooko.

[5] Andreas Loibl. Namecoin. https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2014-08-1/NET-2014-08-1_14.pdf.

[6] bitFlyer. FAQ. What is an altcoin? https://bitflyer.com/en-eu/faq/55-7

[7] H. Kalodner et al. An Empirical Study of Namecoin and Lessons for Decentralized Namespace Design. https://www.semanticscholar.org/paper/An-Empirical-Study-of-Namecoin-and-Lessons-for-Kalodner-Carlsten/5f2f9116d34f741b714b47c4d153566fa77ebe19.

[8] Catalin Cimpanu. OpenNIC drops support for .bit domain names after rampant malware abuse. https://www.zdnet.com/article/opennic-drops-support-for-bit-domain-names-after-rampant-malware-abuse/.

[9] Namecoin Forum. [ANN] NameID - Use namecoin id/ to log into OpenID sites. https://forum.namecoin.org/viewtopic.php?f=2&t=1013.

[10] Kristina Yasuda et al. OpenID for Verifiable Credentials. https://openid.net/wordpress-content/uploads/2022/06/OIDF-Whitepaper_OpenID-for-Verifiable-Credentials-V2_2022-06-23.pdf.

[11] Christopher Allen et al. Decentralized Public Key Infrastructure. https://github.com/WebOfTrustInfo/rwot1-sf/raw/master/final-documents/dpki.pdf.

[12] Welcome to the Web of Trust. https://github.com/WebOfTrustInfo/rwot1-sf/raw/master/final-documents/dpki.pdf.

[13] Volodymyr Pavlyshyn. Zooko's Triangle and DIDs. https://volodymyrpavlyshyn.medium.com/zookos-triangle-and-dids-b51efe273b30.

[14] Decentralized Identifiers (DIDs) v1.0. W3C Recommendation 19 July 2022. https://www.w3.org/TR/did-core/.

[15] Christopher Allen et al. BTCR DID Method. Draft Community Group Report, 8 August 2019. https://w3c-ccg.github.io/didm-btcr/.

[16] Wikipedia. Uniform Resource Identifier (URI). https://en.wikipedia.org/wiki/Uniform_Resource_Identifier.

[17] JSON for Linking Data. https://en.wikipedia.org/wiki/Uniform_Resource_Identifier.

[18] Wikipedia. Web of trust. https://en.wikipedia.org/wiki/Web_of_trust

[19] Holber Berg et al. Overcoming the Asymmetry in the Plastics Value Chain with Digital Product Passports. https://www.econstor.eu/bitstream/10419/251914/1/1798057344.pdf

[20] The did:key Method v0.7. A DID Method for Static Cryptographic Keys. Unofficial Draft 02 September 2022. https://w3c-ccg.github.io/did-method-key/

[21] S. Farrell et al. An Internet Attribute Certificate Profile for Authorization. RFC 5755. https://datatracker.ietf.org/doc/html/rfc5755.

[22] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic. https://eprint.iacr.org/2017/212.pdf.

[23] D.J. Bernstein et al. High-speed high-security signatures. https://ed25519.cr.yp.to/ed25519-20110926.pdf

[24] D. J. Bernstein et al. Twisted Edwards Curves. In *Progress in Cryptology* – AFRICACRYPT 2008, pp 389 – 405. https://link.springer.com/chapter/10.1007/978-3-540-68164-9_26

[25] A NORMAL FORM FOR ELLIPTIC CURVES (cf. Part II, Section 4), by Harold M. Edwards. In BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY, Volume 44, Number 3, July 2007, Pages 393–422. https://community.ams.org/journals/bull/2007-44-03/S0273-0979-07-01153-6/S0273-0979-07-01153-6.pdf

[26] A. Langley et al. Elliptic Curves for Security. IETF RFC 7748. https://datatracker.ietf.org/doc/html/rfc7748

[27] Erik Thormarker. On using the same key pair for Ed25519 and an X25519 based KEM. Cryptology ePrint Archive. Internet-Draft. URL: https://eprint.iacr.org/2021/509.pdf.

[28] Nuttawut Kongsuwan. KERI jargon in a nutshell. Part 2: SAID and ACDC. https://medium.com/finema/keri-jargon-in-a-nutshell-part-2-said-and-acdc-de6bc544b95e

[29] Samuel M. Smith. Key Event Receipt Infrastructure (KERI). https://arxiv.org/abs/1907.02143

[30] Dr. Sam Smith et al. The did:keri Method v0.1 - A DID Method for KERI Identifiers. https://identity.foundation/keri/did_methods/

[31] P. Feairheller. The did:keri DID Method. https://weboftrust.github.io/ietf-did-keri/draft-pfeairheller-did-keri.html

[32] https://github.com/decentralized-identity/keri/tree/master/kids

[33] S. Smith. Key Event Receipt Infrastructure (KERI). https://datatracker.ietf.org/doc/html/draft-ssmith-keri

[34] Samuel M. Smith. Self-Addressing IDentifier (SAID) - draft-ssmith-said-03. https://datatracker.ietf.org/doc/draft-ssmith-said/

[35] Samuel M. Smith. Authentic Chained Data Containers (ACDC) - draft-ssmith-acdc-03. https://datatracker.ietf.org/doc/draft-ssmith-acdc/

[36] Interplanetary File System IPFS. https://docs.ipfs.io

[37] Peer DID Method Specification v1.0 Draft. https://identity.foundation/peer-did-method-spec/

[38] R. West et al. Aries RFC 0023: DID Exchange Protocol 1.0. https://github.com/hyperledger/aries-rfcs/blob/main/features/0023-did-exchange/README.md

[39] DIDComm Messaging v2.x Editor's Draft - DIF Ratified Specification. https://identity.foundation/didcomm-messaging/spec/

[40] T. Looker, Ed. JSON Web Message - draft-looker-jwm-01. Expired Internet draft, January 15, 2020. https://datatracker.ietf.org/doc/html/draft-looker-jwm-01

[41] M. Jones et al. JSON Web Signature (JWS). IETF RFC 7515, May 2015. https://datatracker.ietf.org/doc/html/rfc7515

[42] M. Jones et al. JSON Web Encryption (JWE). IETF RFC 7516, May 2015. https://datatracker.ietf.org/doc/html/rfc7516

[43] M. Jones. JSON Web Algorithms (JWA). RFC 7518, May 2015. https://datatracker.ietf.org/doc/html/rfc7518

[44] N. Madden. Public Key Authenticated Encryption for JOSE: ECDH-1PU – draft-madden-jose-ecdh-1pu-04. https://datatracker.ietf.org/doc/html/draft-madden-jose-ecdh-1pu-04

[45] E. Barker et al. Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication 800-56A Revision 3. April 2018. https://doi.org/10.6028/NIST.SP.800-56Ar3