Go to next chapter ›

8 Decentralized identifiers

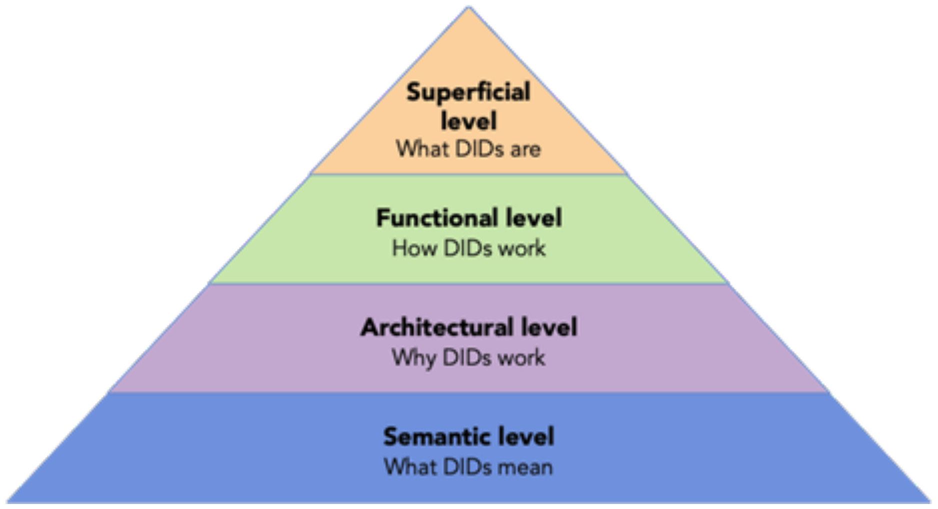
by Drummond Reed and Markus Sabadello

Decentralized identifiers (abbreviated as "DIDs"), are the cryptographic counterpart to verifiable credentials (VCs) that together are the "twin pillars" of SSI architecture. In this chapter you will learn how DIDs evolved from the work started with VCs, how they are related to URLs and URNs, why a new type of cryptographically-verifiable identifier is needed for SSI, and how DIDs are being standardized at World Wide Web Consortium (W3C). Your guides will be two of the editors of the W3C Decentralized Identifier 1.0 specification: Markus Sabadello, Founder and CEO of Danube Tech, and Drummond Reed, Chief Trust Officer at Evernym.

At the most basic level, a **decentralized identifier** (**DID**) is simply a new type of globally unique identifier—not that different from the URLs you see in the address bar of your browser. But at a deeper level, DIDs are the atomic building block of a new layer of decentralized digital identity and public key infrastructure (PKI) for the Internet. This decentralized public key infrastructure (DPKI)[213] could eventually have as much impact on global cybersecurity and cyberprivacy as the development of the SSL/TLS protocol[214] for encrypted Web traffic (currently the largest PKI in the world).

This means you can understand DIDs at four progressively deeper levels (figure 8.1):

Figure 8.x: The progression of four levels at which one can understand DIDs—from a basic definition to a deep understanding how and why they work and what they mean for the future of the Internet and the Web



In this chapter we will travel all the way down through these four levels to provide a much deeper understanding of DIDs.

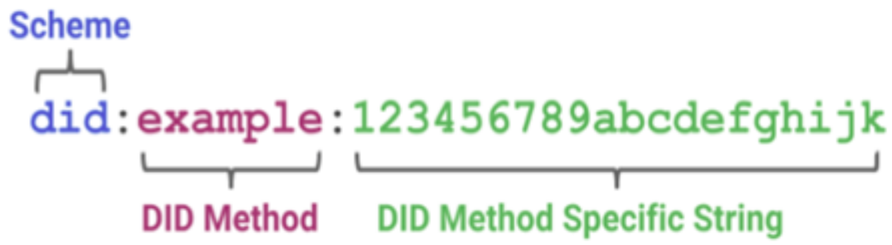
8.1 The superficial level: what is a DID?

To begin with, as defined by the W3C DID specification[215] published by the W3C DID Working Group[216], a DID is a new type of **identifier**: a string of characters that identifies a **resource**[217]. A resource is literally ***anything that can be identified***, from a web page to a person to a planet. This string of characters looks very much like any other Web address, except it doesn't begin with "http:" or "https:", but with "did:".

MEAP

Figure 8.x: The general format of a DID—the scheme name followed by a DID method name followed by the method-

specific string, the syntax for which is defined by the DID method



8.1.1 URIs

In terms of technical standards, a DID is a type of URI (Uniform Resource Identifier). URIs are an IETF standard (RFC 3986)[218] that was adopted by the W3C for identifying any type of resource on the World Wide Web. A URI is a string of characters in a specific format that makes the string globally unique, in other words, no other resource has that same identifier. This of course is very different than human names where many people can have exactly the same name.

8.1.2 URLs

A URL (Uniform Resource Locator) is a URI that can be used to **locate** a **representation** of that resource on the Web. A representation is anything that describes the resource. For example, for a website URL, the resource is a specific page on that site. But if the resource is a person, then he or she obviously can't be "on the Web" directly. So the representation needs to be something describing that person, e.g., a résumé, a blog, a LinkedIn profile, and so on.

Every single resource representation available on the Web has a URL—web page, file, image, video, database record, and so on.

It's not "on the Web" if it doesn't have a URL. The addresses that appear in the address bar of your browser are generally URLs.

Figure 8.x: An example of a browser address bar displaying the URL for the web page about this book

Why do we need the distinction between URIs and URLs if everything on the Web has a URL? Because we also need to identify resources that are **not** on the Web. People, places, planets—all the things for which we had names long before the Internet even existed. All of these are resources that we often need to **refer to** on the Web without the resource actually being **represented on** the Web. A URI that is not a URL is often called an **abstract identifier**.

NOTE

The question of whether a DID by itself serves as a URL—while sounding quite trivial—is actually quite deep. We will have to go all the way down to level four before we can answer it completely.

8.1.3 URNs

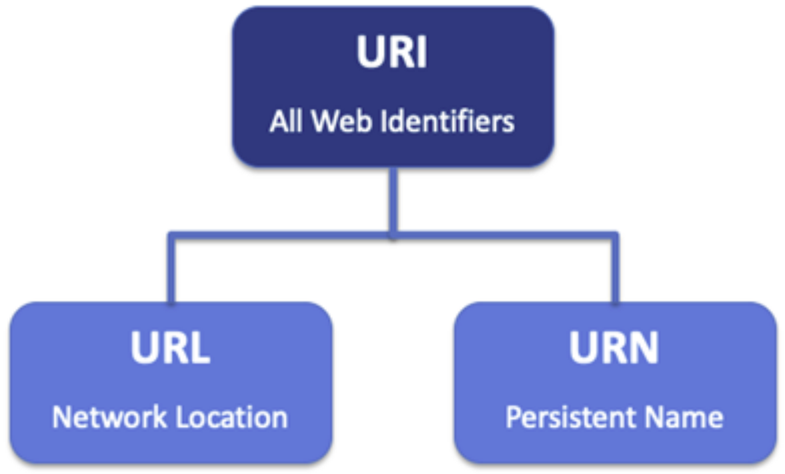
If URLs are the subclass of URIs that point to *the location of a representation of a resource* of the network—which can always change—then what about the subclass of URIs that *identify the* *abstract resource itself* and thus are designed to never change. It turns out there are many uses for this kind of **persistent identifier** (also called a **permanent identifier**). This is exactly what you want when you need to be able to refer to the resource:

- Independent of any particular representation.
- Independent of any particular location.
- Independent of any particular human language or name.
- In a way that will not change over time.

In Web architecture the subclass of URIs reserved for persistent identifiers are called *Uniform Resource Names* (URNs). They are defined by RFC 8141[219], which goes into great detail about both the syntax and the policies necessary to manage namespaces for identifiers that are meant to never change. (Think about how complex it would get if every phone number, email address, and human name could be assigned only once and never reused for another person ever again for the rest of time.)

Figure 8.x shows how URLs and URNs are both subclasses of URIs.

Figure 8.x: URLs and URNs are both subclasses of URIs—URLs always point to representations of resources on the Web, whereas URNs are persistent names for any resource, on or off the Web

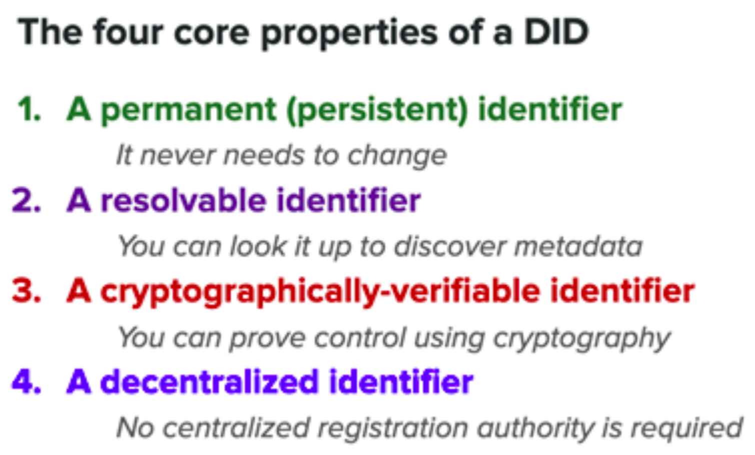


8.1.4 DIDs

Having described URIs, URLs, and URNs, we can now be more precise about the definition of a DID: a DID is a URN that can be looked up (“resolved”) to get a standardized set of information (“metadata”) about the resource identified by the DID (as described in the next section of this chapter). If the identified resource has one or more representations on the Web, the metadata can include one or more of those URLs.

But that definition captures only two of the four properties of a DID—**persistence** and **resolvability**. It is the other two properties—**cryptographic verifiability** and **decentralization**—that most strongly distinguish a DID from other URIs or any other globally unique identifiers. At the first meeting of the W3C DID Working Group in September 2019 at Fukuoka, Japan, one presenter summarized the four properties this way:

Figure 8.x: A summary of the four core properties of a DID presented at the first meeting of the W3C DID Working Group



The four core properties of a DID

1. **A permanent (persistent) identifier**
   It never needs to change
2. **A resolvable identifier**
   You can look it up to discover metadata
3. **A cryptographically-verifiable identifier**
   You can prove control using cryptography
4. **A decentralized identifier**
   No centralized registration authority is required

What is special about the third and fourth properties is that they both depend on cryptography. In the first case, cryptographic verifiability, cryptography is used to generate the DID. Since the DID is now associated with exactly one public/private key pair, the controller of the private key can prove that he/she/it is also the controller of the DID. (See *The Architectural Level* section of this chapter for more details.)

In the second case, decentralized registration, cryptography is what eliminates the need for centralized registration authorities—the kind that are needed for almost every other globally unique identifier we use, from postal addresses to telephone numbers to domain names. It is the centralized registries run by these authorities that can determine if a particular identifier is unique—and allow it to be registered only if it is.

By contrast, cryptographic algorithms for public/private key pairs are based on random number generators, large prime numbers, elliptic curves, or other cryptographic techniques for producing globally unique values that do not require a central registry to effectively guarantee uniqueness. We say "effectively guarantee" because there is an infinitesimally small chance of a collision[220] with someone else using the same algorithm. But this chance of collision is mathematically so small that for all practical purposes it can be ignored.

As a result, anyone with the proper software can generate a DID according to a particular DID method (discussed in the following sections) and begin using it immediately without requiring the authorization or involvement of any centralized registration authority. This is the same process used to create public addresses on the Bitcoin or any other popular blockchain—it is the essence of what makes a DID ***decentralized***.

8.2 The functional level: how DIDs work

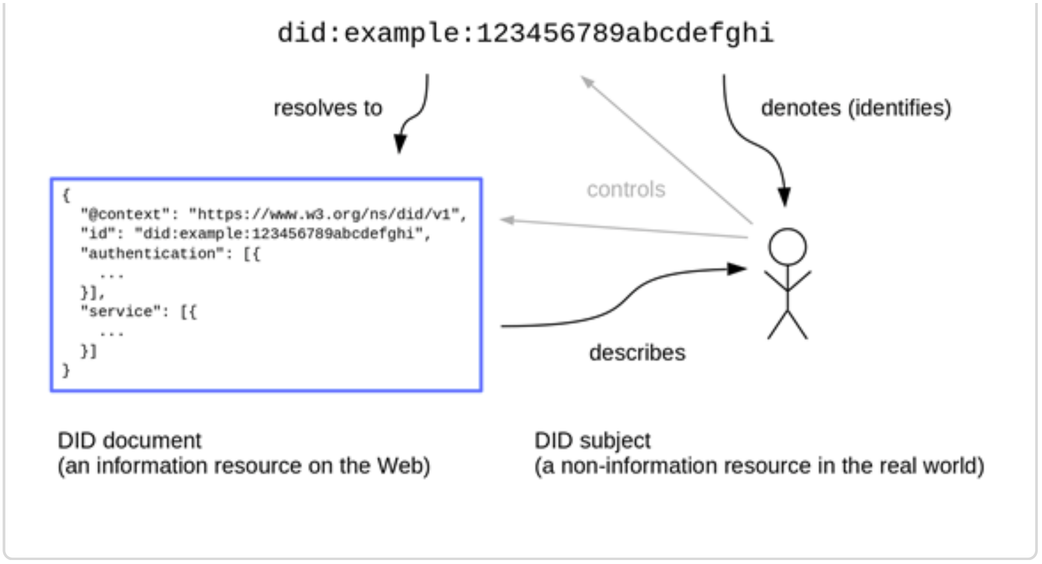
8.2.1 DID documents

Identifiers can of course be useful just by themselves, as strings of text characters that can be used to refer to a resource. This string can be stored in a database or in a document, or it could be printed on a t-shirt or business card, or attached to an email. But for digital identifiers, the usefulness comes just not from the identifier itself, but from how it can be used by applications designed to consume that particular type of identifier. For example, a typical Web address that starts with "http" or "https" is not very interesting as a string by itself. It only becomes useful once you type it into a Web browser or click on a hyperlink in order to access a representation of the resource behind the identifier (such as a Web page).

This is similar with DIDs: Although (at least today) it is not possible to type a DID into a Web browser, you can give it to a specialized piece of software (or hardware) called a **DID resolver** that will use it to retrieve a standardized data structure called a **DID document**. This data structure is not like a Web page or an image file, in other words it is not designed to be viewed directly by end users in a Web browser or similar software.[221] Instead, it is consumed by digital identity applications or services such as wallets, agents, or personal data stores, all of which use DIDs as fundamental building blocks.

Every DID has exactly one associated DID document. The DID document contains metadata about the **DID subject**, which is the term for the resource **identified by** the DID and **described by** the DID document. For example, a DID for a person (the DID subject) has an associated DID document that contains metadata about that person. The entity that controls the DID and its associated DID document is called the **DID controller**. In many cases, this is the same as the **DID subject**, but they could also be different. An example is when a parent controls a DID that identifies their child—the DID subject is the child but the DID controller (at least until the child comes of age) is the parent.

Figure 8.x: Relationships between the DID, DID document, and DID subject (in cases where the DID subject is also the DID controller)


did:example:123456789abcdefghi
resolves to
denotes (identifies)
controls
describes
DID document
(an information resource on the Web)
DID subject
(a non-information resource in the real world)

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    ...
  }],
  "service": [{
    ...
  }]
}
```

Theoretically, a DID document can contain any arbitrary information about the DID subject, even personal attributes such as a name or an e-mail address. In practice however, this is problematic for privacy reasons; instead, the recommended best practice is for a DID document to contain only the minimum amount of machine-readable metadata required in order to enable **trustable interaction** with the DID subject. Typically, this includes the following:

- One or more **public keys** (or other verification methods) that can be used to authenticate the DID subject during an interaction. This is what makes interactions involving DIDs trustable, and this is also the essence of the DPKI enabled by DIDs.
- One or more **services** associated with the DID subject that can be used for concrete interaction via protocols supported by those services. This can include a wide range of protocols from instant messaging and social networking, to dedicated identity protocols such as OpenID Connect (OIDC), DIDComm as described in chapter 5 about SSI architecture, and others.
- Certain additional metadata such as **timestamps**,

digital signatures and other **cryptographic proofs**, or metadata related to **delegation and authorization**.

Figure 8.x is an example of a very simple DID document using a JSON-LD encoding.

Figure 8.x: Example DID document with one public key used for authentication, and one service

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
      "id": "did:example:123456789abcdefghi#keys-1",
      "type": "Ed25519VerificationKey2018",
      "controller": "did:example:123456789abcdefghi",
      "publicKeyBase58" : "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZ
  }],
  "service": [{
      "id":"did:example:123456789abcdefghi#vcs",
      "type": "VerifiableCredentialService",
      "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

copy

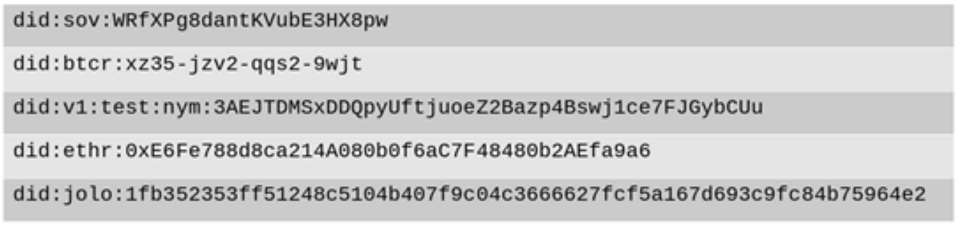
This metadata that is associated with every DID, especially public keys and services, is the technical basis for all interaction that happens between different actors in an SSI ecosystem.

8.2.2 DID methods

As we explained in the previous sections, DIDs are not created and maintained in a single type of database or network like most other types of URIs. There is no authoritative centralized registry—or a hierarchy of federated registries like DNS—where all DIDs are written and read. In fact, many different types of DIDs exist in today's SSI community—see the ***Types of DIDs*** section later in this chapter. They all support the same basic functionality but they differ in how that functionality is implemented, e.g. how exactly a DID is created, or where and how a DID's associated DID document is stored and retrieved.

These types or classes of DIDs are known as **DID methods**. The second part of the DID identifier format—between the first and second colons —is called the **DID method name**. Figure 8.x shows examples of DIDs created using five different DID methods: sov (Sovrin), btcr (Bitcoin), v1 (Veres One), ethr (Ethereum), and jolo (Jolocom).

Figure 8.x: Example DIDs generated using five different DID methods



```
did:sov:WRfXPg8dantKVubE3HX8pw
did:btcr:xz35-jzv2-qqs2-9wjt
did:v1:test:nym:3AEJTDMSxDDQpyUftjuoeZ2Bazp4Bswj1ce7FJGybCUu
did:ethr:0xE6Fe788d8ca214A080b0f6aC7F48480b2AEfa9a6
did:jolo:1fb352353ff51248c5104b407f9c04c3666627fcf5a167d693c9fc84b75964e2
```

As of the writing of this book, more than 40 DID method names have been registered at the informal DID Method Registry[222] maintained by the W3C Credentials Community Group. Each DID method is required to have its own technical specification, which must define the following aspects of the DID method:

- The syntax of the third part of the DID identifier format (the portion after the second colon). This is called the **method-specific identifier** and is typically a long string which is generated using random numbers and cryptographic functions. It is always guaranteed to be unique within the DID method namespace.

- The four basic operations that can be executed on a DID:
- o **Create:** How can a DID and its associated DID document be created?
- o **Read:** How can the associated DID document be retrieved?
- o **Update:** How can the contents of the DID document be changed?
- o **Deactivate:** How can a DID be deactivated so it can no longer be used?
- Security and privacy considerations specific to the DID method.

It is difficult to make generic statements about the four DID operations, since DID methods can be designed in very different ways. For example, some DID methods are based on blockchains or other distributed ledgers. In this case, creating or updating a DID typically involves writing a transaction to that ledger. Other DID methods do not use a blockchain; they implement the four DID operations in other ways (see ***Types of DIDs*** later in this chapter).

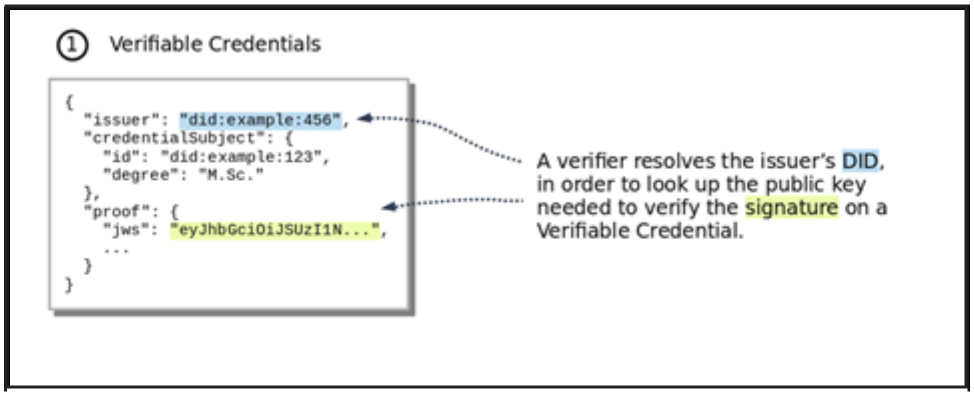
One consequence of the technological variety of DID methods is that some may be better suited for certain use cases than others. DID methods may differ in how "decentralized" or "trusted" they are, as well as in factors of scalability, performance, or cost of the underlying technical infrastructure. The charter of the W3C DID Working Group[223] includes a deliverable of a “rubric” document to help evaluate how well a particular DID method will meet the needs of a particular user or community.

8.2.3 DID resolution

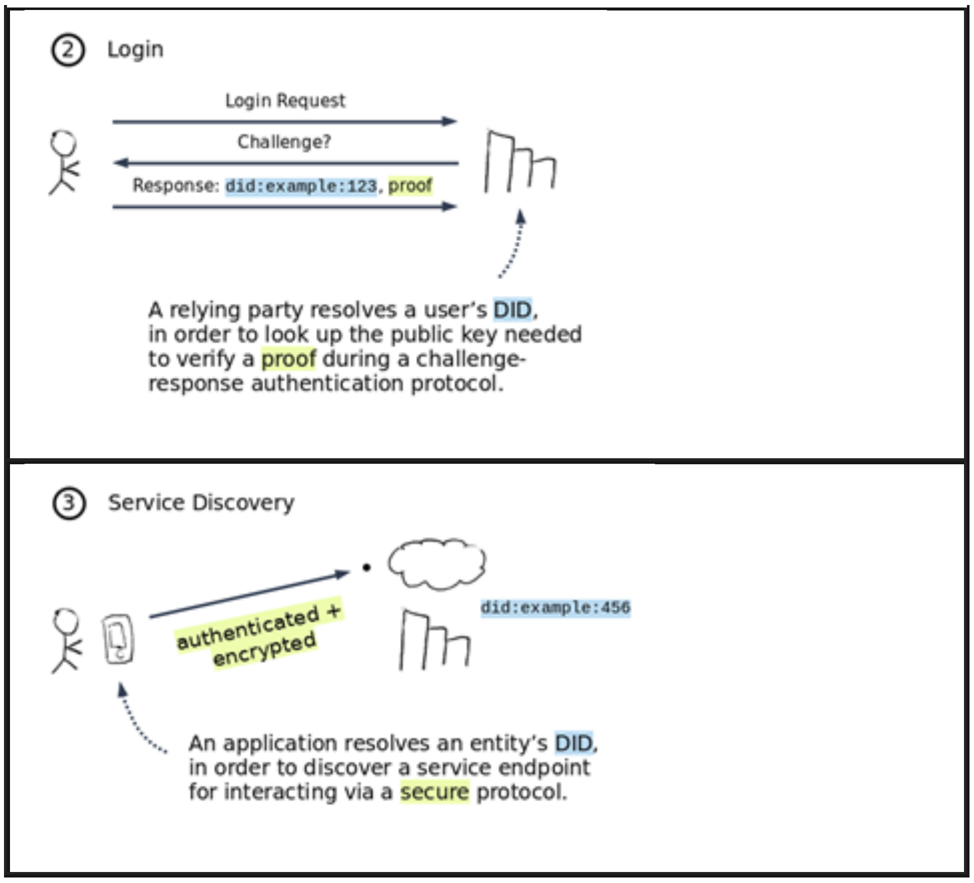
The process of obtaining the DID document associated with a DID is called ***DID resolution***. This process allows DID-enabled applications and services to discover the machine-readable metadata about the DID subject that is expressed by the DID document. This metadata can be used for further interaction with the DID subject, for example:

- To look up a public key in order to verify a digital signature from an Issuer of a Verifiable Credential.
- To authenticate the DID controller when he/she needs to "log in" to a website or app.
- To discover and access a well-known service associated with the DID controller, such as a website, social network, or licensing authority.
- To request a DID-to-DID connection with the DID controller.

Figure 8.x: Common scenarios requiring DID resolution—the first one is using a DID to identify the issuer of a verifiable credential; the second to login to a website, and the third to discover a service associated with the DID


① Verifiable Credentials
A verifier resolves the issuer's DID, in order to look up the public key needed to verify the signature on a Verifiable Credential.

```
{
  "issuer": "did:example:456",
  "credentialSubject": {
    "id": "did:example:123",
    "degree": "M.Sc."
  },
  "proof": {
    "jws": "eyJhbGciOiJSUzI1N...",
    ...
  }
}
```


② Login
Login Request
Challenge?
Response: did:example:123, proof
A relying party resolves a user's DID, in order to look up the public key needed to verify a proof during a challenge-response authentication protocol.
③ Service Discovery
authenticated + encrypted
did:example:456
An application resolves an entity's DID, in order to discover a service endpoint for interacting via a secure protocol.

The DID resolution process is based directly on the Read operation defined by the applicable DID method, which as noted can vary considerably depending on how the DID method is designed. This means DID resolution is not confined to a single protocol in the same way as DNS (which makes it possible to resolve domain names to IP addresses) or HTTP (which is used to retrieve representations of resources from Web servers).

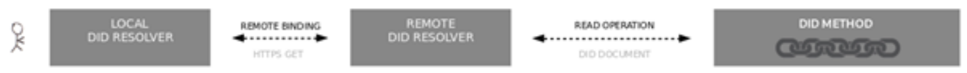
For example, no assumption should be made that a blockchain, distributed ledger, or database (centralized or decentralized) is used for resolving DIDs, or even that interaction with a remote network is required during the DID resolution process. Furthermore, a DID document is not necessarily stored in plain-text in a database or available for download from a server.

Though some DID methods may work like this, others may define more complex DID resolution processes that involve on-the-fly construction of a "virtual" DID document.

Therefore, rather than thinking of DID resolution as a protocol, it should be considered an abstract function or algorithm, which takes a DID (plus optional additional parameters) as its input, and returns the DID document (plus optional additional metadata) as its result.

DID resolvers can come in several architectural forms: they can be implemented as a native library included in an application or even an operating system, in the same way as DNS resolvers are included in all modern operating systems. Alternatively, a DID resolver can be provided by a third party as a hosted service, responding to DID resolution requests via HTTP or other protocols (called *bindings*). Mixed forms are also possible. For example a local DID resolver could delegate part of or all of the DID resolution process to a pre-configured remotely hosted DID resolver. This is similar to how DNS resolvers in our local operating systems typically query a remote DNS resolver hosted by an Internet Service Provider (ISP), which performs the actual DNS resolution work.

Figure 8.x: An example of a local DID resolver querying a remote DID resolver, which then retrieves the DID document according to the applicable DID method

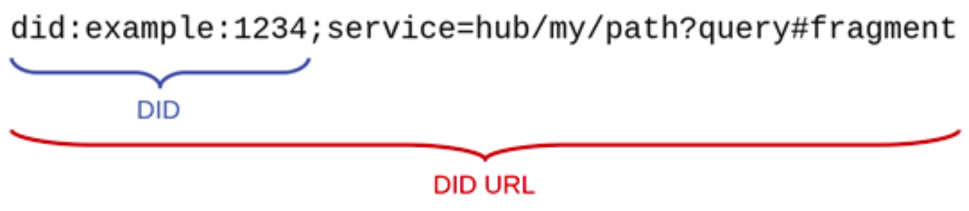


Of course reliance on an intermediary service during the DID resolution process introduces potential security risks and potential elements of centralization, both of which can impact the security and trust properties of other layers in the SSI stack that rely on DIDs. Therefore whenever possible DID resolution should be integrated directly into a DID-enabled application in such a way that the application is able to independently verify that the DID resolution result is correct (meaning the correct DID document has been obtained).

8.2.4 DID URLs

DIDs are powerful identifiers by themselves, but they can also be used as the basis for constructing more advanced URLs rooted in a DID. This is similar to how http and https URLs can consist not only of a domain name, but also have other syntactic components appended to the domain name: an optional path, an optional query string, and an optional fragment. With URLs, these enable identifying arbitrary resources under the authority of the domain name. The same is true for DIDs. This means that, although the most important function of a DID is to resolve it to a DID document, it can also serve as a root authority of a set of **DID URLs** that enable an "identifier space" for additional resources associated with the DID.

Figure 8.x: DIDs and DID URLs



DID URLs can be used for many different purposes. Some uses of

DID URLs will be well-known and standardized, whereas others are completely extensible and DID method- or application-dependent. See Table 8.1 for example DID URLs and their meanings:

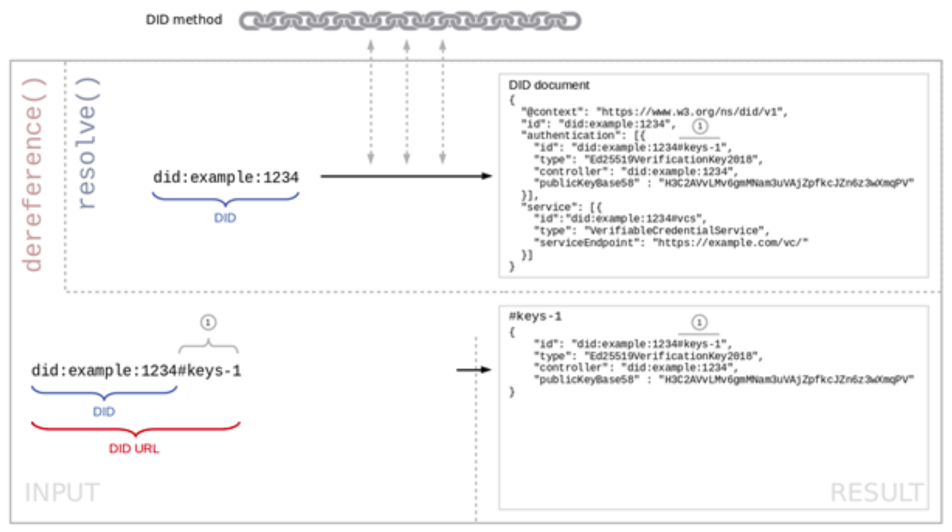
Table 8.1: Example DID URLs and their meanings

`did:example:1234/`	The simplest possible DID URL that is just the DID itself plus a forward slash to identify the DID document. See the very last section of this chapter for more about this.
`did:example:1234#keys-1`	DID URL with a fragment. This DID URL identifies a specific public key inside the DID's associated DID document. This is similar to how an "http" or "https" URL with a fragment can point to a specific part of an HTML Web page.
`did:example:1234;version-id=4`	DID URL with a DID parameter ("version-id"). This DID URL identifies a previous version of the DID's associated DID document, as opposed to the latest version. This is useful in situations when the contents of a DID document have been updated, but a stable reference to a specific version is needed.
`did:example:1234;version-id=4#keys-1`	Combination of the previous two examples. This DID URL identifies a specific public key inside a specific previous version of the DID's associated DID document.
`did:example:1234/my/path?query#fragment`	DID URL with additional syntactic components (path,

query string, fragment). The meaning and processing rules of this DID URL are not defined by the core DID standard, but are dependent on the DID method and/or the applications that use these identifiers.	
`did:example:1234;service=hub/my/path?query#fragment` DID URL with a DID parameter ("service"). This DID URL identifies a specific service inside the DID's associated DID document (in this case, the "hub" service). The meaning and processing rules of the remaining syntactic components (path, query string, fragment) are not defined by the core DID standard, but are specific to the "hub" service.	

Once a DID document has been retreived by a DID resolver—the process called DID ***resolution***—for most DID URLs there is a second stage called DID ***dereferencing***. Whereas resolution only returns the DID document, in the dereferencing stage, the DID document is further processed to accessing or retrieving the resource identified by the DID URL (which almost always is a ***different resource*** than the DID subject). These two terms—resolution and dereferencing—are defined by the URI standard (RFC 3986), and they apply not only to DIDs, but to all types of URIs and the resources they identify. Figure 8.10 is an example of how these two processing stages differ.

Figure 8.x: The process of first resolving a DID and then dereferencing a DID URL containing a fragment; the result is a specific public key inside the DID document



```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:1234",
  "authentication": [{
    "id": "did:example:1234#keys-1",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:1234",
    "publicKeyBase58" : "H3C2AVvLMv6gmMNam3uVAjZpfkcJZn6z3wXmqPV"
  }],
  "service": [{
    "id":"did:example:1234#vcs",
    "type": "VerifiableCredentialService",
    "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

```
{
    "id": "did:example:1234#keys-1",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:1234",
    "publicKeyBase58" : "H3C2AVvLMv6gmMNam3uVAjZpfkcJZn6z3wXmqPV"
}
```

8.2.5 Comparison with the Domain Name System (DNS)

We have already used domain names and the domain name system (DNS) as an analogy when explaining certain aspects of DIDs and how they are different from other identifiers. The following table summarizes the similarities and differences between DIDs and domain names.

Table 8.x Comparison of DIDs with domain names

Decentralized Identifiers (DIDs)	Domain Names
Globally unique	Globally unique
Persistent	Reassignable
Machine-friendly identifiers (i.e., long character strings based on	Human-readable names

random numbers and cryptography)	
Resolvable using different mechanisms defined by the applicable DID method	Resolvable using the standard DNS protocol
Associated data is expressed in DID documents	Associated data is expressed in DNS zone files
Fully decentralized namespace without delegation	Hierarchical, delegatable namespaces based on centralized root registries of top-level domain names (TLDs)
Secured by DID method-specific processes and infrastructure (e.g. blockchains)	Secured by trusted root registries and traditional PKI (DNSSEC)[224]
Cryptographically-verifiable	Verifiable using DNS security extensions (DNSSEC)
Used as authority component in DID URLs	Used as authority component in "http" and "https" Web addresses as well as e-mail addresses and other identifiers

Governed by the authority for each DID method (anyone can create a DID method)	Governed by the Internet Corporation for Assigned Names and Numbers (ICANN)
Fully under the control of the DID controller	Ultimately controlled by ICANN and the registry operator for each DNS TLD

8.2.6 Comparison with Uniform Resource Names (URNs) and other Persistent Identifiers

As we explained previously, DIDs meet the functional requirements of URNs, i.e., they are persistent identifiers that always identify the same entity and cannot be reassigned. There are many other types of persistent identifiers that differ from DIDs in ways that make them generally less suitable for SSI applications. The following table lists other types of persistent identifiers and how they compare to DIDs.

Table 8.x: Comparison of DIDs with other types of persistent identifiers

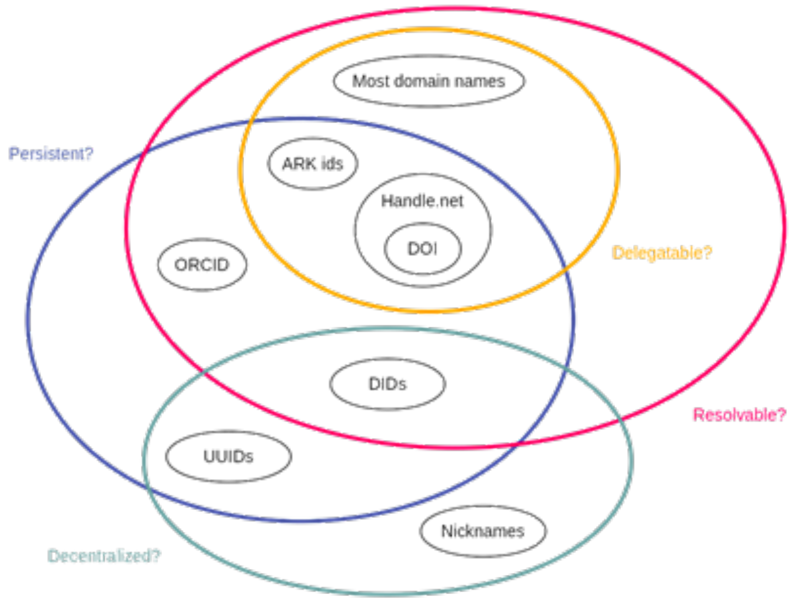
Other URNs	· Either not resolvable, or the resolution process and metadata varies with each type · Not cryptographically verifiable

Universally Unique Identifiers (UUIDs, also called Globally Unique Identifiers or GUIDs)	· Not resolvable · Not cryptographically verifiable
Persistent URLs (PURLs) Handle System (HDLs) Digital Object Identifiers (DOIs), Archival Resource Keys (ARKs), Open Researcher and Contributor ID (ORCID)	· Not decentralized, i.e. creating and using these identifiers depends on a central or hierarchical authority · Not cryptographically verifiable

MEAP

Figure 8.x: Compared to other identifiers, DIDs are persistent, resolvable, and decentralized (cryptographic verifiability is not shown since none it does not apply to any of the other identifiers)

Figure 8.11 is a visual way of depicting how most of the other URIs in use today compare to DIDs. Note that it does not include a circle for cryptographically verifiability **because DIDs are the only identifier that offers that property. However it does include a circle for one property DIDs do** not **have:** delegatability**. That is the ability for one identifier authority to delegate a subnamespace to another identifier authority. An example is a domain name like maps.google.com, where the .com registry delegates to Google and Google delegates to its maps service.**



MEAP

8.2.7 Types of DIDs

Since their invention, interest in DIDs has grown exponentially (for reasons we explain in much greater detail in the next section). Although the first DID methods were closely tied to blockchains and distributed ledgers, as DIDs started evolving, many more types of DIDs have been developed. A series of presentations at the first meeting of the W3C DID Working Group in September 2019 in Fukuoka, Japan, described the various DID methods that have been developed at that point as falling into the broad categories described in Table 8.4.

Table 8.x: The broad categories of DID methods that have been developed as of September 2019

Category	Description & Examples

Ledger-based DIDs	The "original" category of DID methods involves a blockchain or other distributed ledger technology (DLT), which serves the purpose of a registry that is not controlled by a single authority. This registry is typically public and globally accessible. A DID is created/updated/ deactivated by writing a transaction to the ledger, which is signed with the private key of the DID controller.
	
	`did:sov:WRfXPg8dantKVubE3HX8pw`
	`did:btcr:xz35-jzv2-qqs2-9wjt`
	`did:ethr:0xE6Fe788d8ca214A080b0f6aC7F48480b2AEfa9a6`
	`did:v1:test:nym:3AEJTDMSxDDQpyUftjuoeZ2Bazp4Bswj1ce7FJGybCUu`

Ledger Middleware **("Layer 2") DIDs**	An improvement to classic ledger-based DID methods, this category adds an additional storage layer such as a distributed hash table (DHT) or traditional replicated database system "on top" of the base layer blockchain. DIDs can be created/updated/deactivated at this second

	layer without requiring a base layer ledger transaction every time. Instead, multiple DID operations are batched into a single ledger transaction, increasing performance and decreasing cost.
	
	`did:ion:test:EiDk2RpPVuC4wNANUTn_4YXJczjzi10zLG1XE4AjkcGOLA`
	`did:elem:EiB9htZdL3stukrklAnJ0hrWuCdXwR27TNDO7Fh9HGWDGg`

Peer DIDs	This is a special category for a DID method that does not require a globally shared registration layer such as a blockchain. Instead, a DID is created and subsequently exists only within a relationship between a limited number of participants. The DIDs that are part of the relationship are exchanged via a peer-to-peer protocol, resulting in private connections between the participants. `did:peer:1zQmZMygzYqNwU6Uhmewx5Xepf2VLp5S4HLSwwgf2aiKZuwa`
Static DIDs	There is also a category of DID methods that are "static", i.e. they can only be created and resolved, but not updated or deactivated. Such DID methods tend to not require complex protocols or storage infrastructure. For example, a DID may simply be a "wrapped" public key, from which an entire DID document can be resolved algorithmically, without requiring any data other than the DID itself. `did:key:z6Mkfriq1MqLBoPWecGoDLjguo1sB9brj6wT3qZ5BxkKpuP6`
Alternative DIDs	A number of other innovative DID methods have been developed that do not fall into any of the previous categories. They

MEAP

	demonstrate that the DID concept is flexible enough to be layered on top of existing Internet protocols, such as Git, the Interplanetary File System (IPFS), or even the Web itself.
	
	`did:git:625557b5a9cdf399205820a2a716da897e2f9657`
	`did:ipid:QmYA7p467t4BGgBL4NmyHtsXMoPrYH9b3kSG6dbgFYskJm`
	`did:web:uport.me`

8.3 The architectural level: why DIDs work

Having explained what DIDs are and how they work, let's now go a step deeper into *why* they work. What is there so much interest in this new type of identifier?

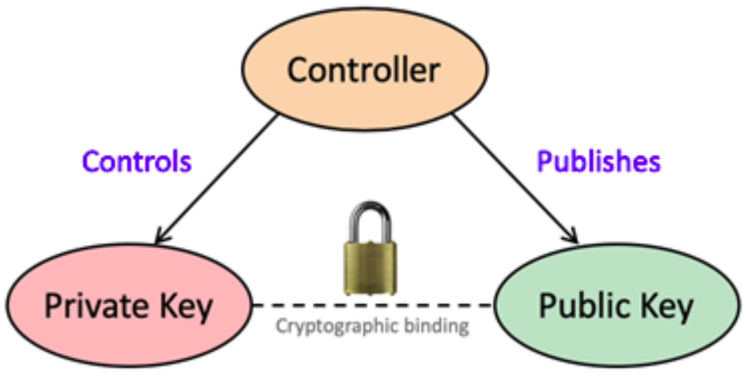
To answer this, we must delve more deeply into the core problems they solve, which are less problems of identity and more problems of *cryptography*.

8.3.1 The core problem of Public Key Infrastructure (PKI)

Since it was first conceived, PKI has one hard problem at its very core. It is not a problem with cryptography per se, i.e., with the math involved with public/private keys or encryption/decryption algorithms. Rather it is a problem with *cryptographic infrastructure*, i.e., how we can make public/private key cryptography easy and safe for people and organizations to use at scale.

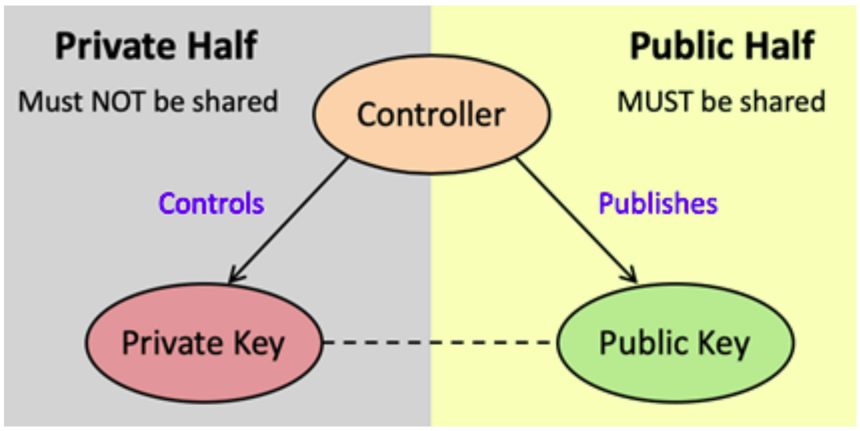
This is not an easy problem—in fact it has vexed PKI ever since the term was invented. The reason lies in the very nature of how public/private key cryptography works. To understand this, let's take a look at the basic PKI "trust triangle" (figure 8.x). It shows that it's not enough to think about public/private "key pairs". You have to see each key pair in relation to its controlling authority ("controller"), be that a person, an organization, or even a thing (if the thing has the capacity to generate key pairs and store them in a digital wallet).

Figure 8.x: The basic trust triangle at the heart of all public/private key cryptography



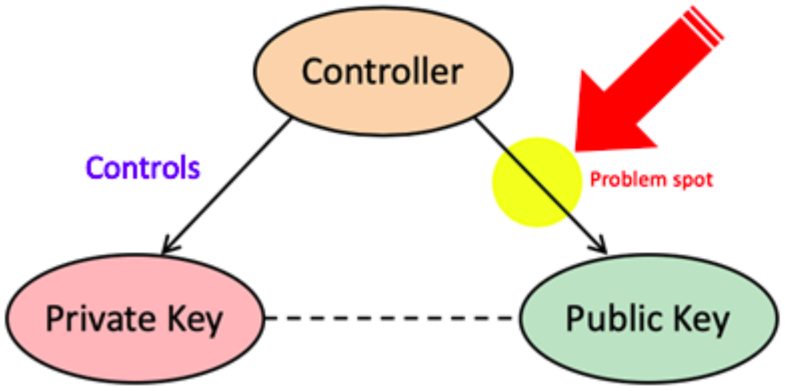
Public and private keys are bound to each other mathematically such that neither can be forged—each can be used only for a specific set of functions defined by a specific cryptographic algorithm. But *both* types of keys are intrinsically related to the controller. Regardless of the algorithm, these two fundamental roles are highlighted in figure 8.x. The private key must be reserved for the exclusive use of the controller (or its delegates) and must never be revealed to anyone else. By contrast the public key is just the opposite: it **must** be shared with any party that wishes to securely communicate with the controller. It is the only way to encrypt messages to—or verify messages from—that controller.

Figure 8.x: The fundamental roles of public keys vs. private keys in PKI



Although the task of keeping a private key private is not trivial by any means, that is not the hard problem at the heart of PKI. Rather the hard problem is on the *other* side of the PKI trust triangle as shown in figure 8.14.

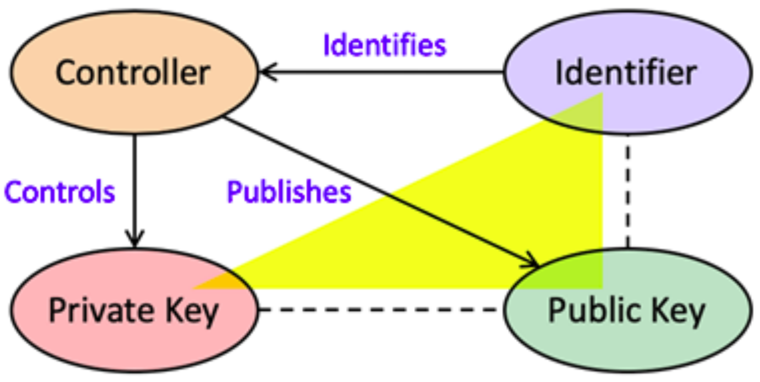
Figure 8.14: The core problem at the heart of PKI: how do you bind a public key to its controller?



The problem is simply this: ***how do you strongly bind a public key to its controller*** so that any party relying on that public key (the **relying party**) can be sure they are dealing with the real controller? After all, if you can fool a relying party into accepting the public key for controller B when the relying party thinks it is the public key for controller A, then for all intents and purposes controller B can fully impersonate controller A. The cryptography will all work perfectly and the relying party will never know the difference—until they become the victim of whatever cybercrime controller B is perpetrating.

Therefore, as a relying party, it is essential to know ***you have the correct public key at the correct point in time*** for any controller you are dealing with. And that indeed is a hard problem because, whereas public keys are purely digital entities whose cryptographic validity can be verified in milliseconds, controllers are **not**. They are real people, organizations, or things that exist in the real world. So the only way to digitally bind a public key to a controller is to add one more piece of the puzzle: a digital dentifier for the controller.

Figure 8.15: The real **PKI trust triangle includes a digital identifier for the controller**



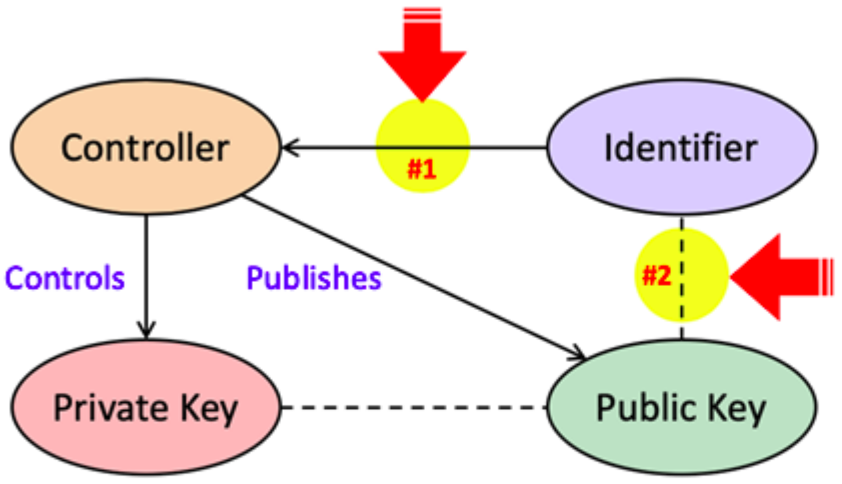
The *real* PKI trust triangle is the yellow triangle shown in figure 8.15. The new piece is an identifier for the controller that can be bound to the public key in such a way that relying parties can be confident the public key belongs to the controller and nobody else.

What figure 8.15 reveals is that this identifier-binding problem actually has two parts:

1. How do you strongly bind the identifier to the controller?
2. How do you strongly bind the public key to the identifier?

These two problem spots are called out in figure 8.16:

Figure 8.16: The two problem spots when it comes to strongly binding an identifier to a controller



These are the two problem areas that PKI has struggled with since it was born in the 1970s. In the balance of this section we'll look at four different solutions to these two problems:

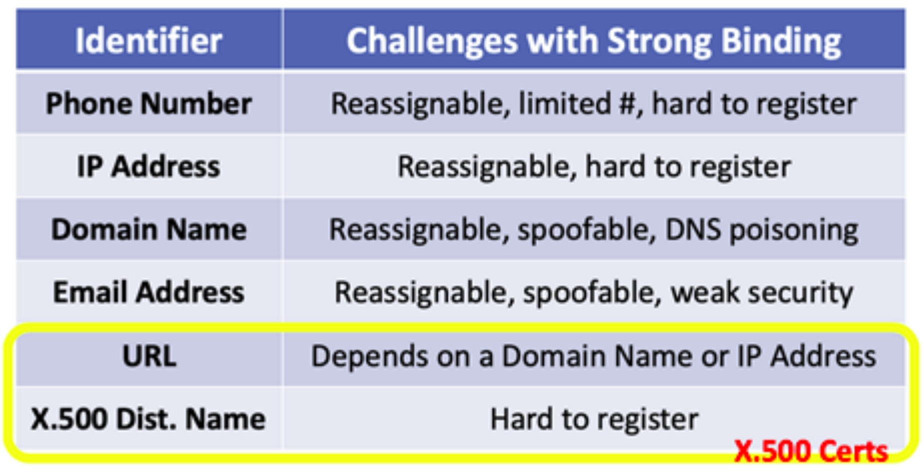
1. The conventional PKI model
2. The web-of-trust model
3. Self-certifying identifiers
4. DIDs

8.3.2 Solution #1: The conventional PKI model

The first solution is the conventional PKI model for issuing **digital certificates** ("certs"). This is the predominant model that has evolved over the last 40 years. Probably the best known example is the SSL/TLS PKI that uses X.500 certs to provide secure connections in our browsers using the HTTPS protocol (the lock you see in your browser address bar).

The conventional PKI solution to the first problem—identifier-to-controller binding—is to use one of the most suitable existing identifiers and then follow industry best practices for ensuring its strong binding to a controller. Table 8.5 summarizes the available choices of identifiers and highlights the ones used most often in X.500 certificates.

Table 8.5: The different identifier choices for conventional PKI—highlighted are those typically used in X.500 digital certificates



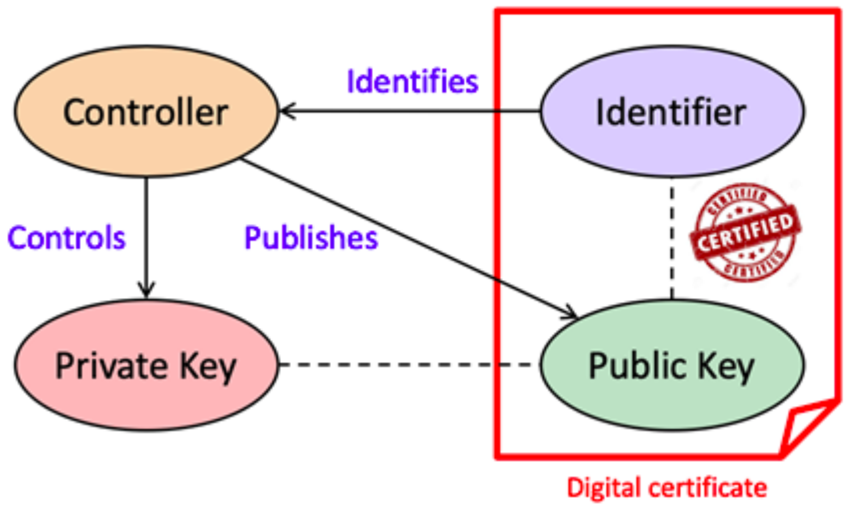
Identifier	Challenges with Strong Binding
Phone Number	Reassignable, limited #, hard to register
IP Address	Reassignable, hard to register
Domain Name	Reassignable, spoofable, DNS poisoning
Email Address	Reassignable, spoofable, weak security
URL	Depends on a Domain Name or IP Address
X.500 Dist. Name	Hard to register

The primary advantage of a URL (based on a domain name or IP address) is that there are automated tests that can be performed to ensure that the controller of the public key also controls the URL. However these tests cannot detect homographic attacks[225] (using look-alike names or look-alike characters from different international alphabets) or DNS poisoning[226].

The primary advantage of an X.500 Distinguished Name (DN) is that it can be administratively verified to belong to the controller. However this verification must be performed manually and thus is always subject to human error. Furthermore, it is not an easy process to register an X.500 DN—certainly not something the average Internet user can be expected to undertake.

The conventional PKI approach to the second problem—public-key-to-identifier binding—seems obvious in the context of cryptography: ***digitally sign a document*** containing both the public key and the identifier. That is the origin of the **public key certificate**[227] (a specific kind of digital certificate). This solution is illustrated in figure 8.17:

Figure 8.17: Conventional PKI solves the public-key-to-identifier binding problem using digital certificates signed by some type of certificate authority



The question, of course, is *who digitally signs this digital certificate*? This introduces the whole notion of a **trusted third party (TTP)**[228] —someone the relying party must trust to sign the digital certificate or else the whole idea of PKI falls apart. The conventional PKI answer is a **certificate authority (CA)**—a service provider whose entire job is to follow a specified set of practices and procedures to confirm the identity of a controller and the authenticity of its public key before issuing a digital certificate that binds the two and signing it with the CA's private key.

Different PKI systems use different certification programs for CAs; one of the best known is the **WebTrust** program originally developed by American Institute of Certified Public Accountants and now run by the Chartered Professional Accountants of Canada[229]. Certification is obviously critical for CAs because acting as a TTP is inherently a human process—it cannot be automated (if it could, a TTP would not be needed). And unfortunately humans make mistakes.

But being human is only one issue with TTPs. Table 8.6 lists the other drawbacks.

Table 8.6: The drawbacks of using conventional PKI to solve the identifier binding problem

Drawbacks of TTPs	Description
Cost	Inserting a TTP (that must perform work only humans can do) into trust relationships adds costs that someone must pay for
Friction	Introducing a TTP requires additional work on the part of all the parties to a trust relationship
Single point of failure	Each TTP becomes an attack point because a single breach can compromise all of its digital certificates
Identifiers changing	If an identifier changes, the old digital certificate must be revoked and a new one issued
Public keys changing	When a public key is rotated (as most security policies require periodically), the old digital certificate must be revoked and a new one issued

Despite all these drawbacks, conventional PKI has so far been the only commercially viable solution to the identifier binding problem. But as the Internet has climbed in usage and commercial value—and rate cybercrime with it—so has the demand for a better solution.

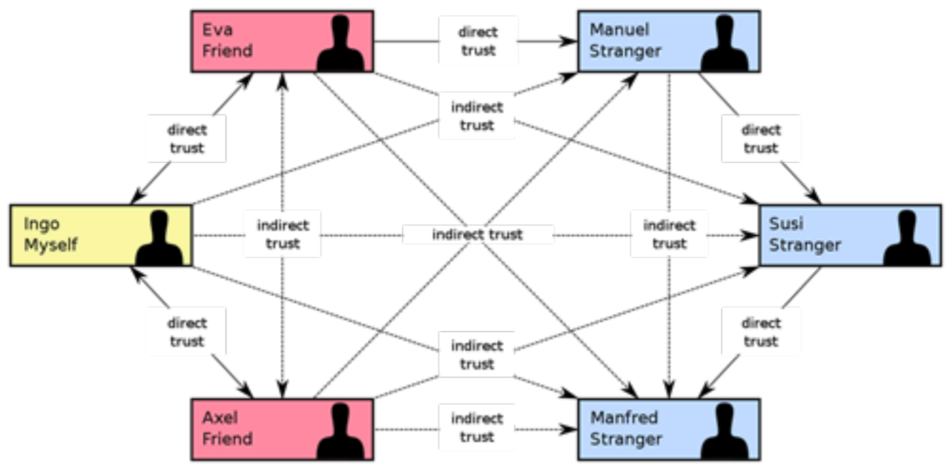
8.3.3 Solution #2: The web-of-trust model

One alternative to the conventional PKI model was formulated early on by one of the pioneers in public/private key cryptography, Phillip Zimmermann, inventor of Pretty Good Privacy (PGP). He coined the term “web of trust” for this model because it didn’t rely on centralized CAs, but rather on individuals who knew each other directly and therefore could individually sign each others public keys—effectively creating ***peer-to-peer digital certificates***. Here’s Mr. Zimmermann’s description of this model from the 1992 manual for PGP version 2.0[230]:

As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys.

Figure 8.18 is a visual depiction of how the web-of-trust model[231] works.

Figure 8.18: A visual diagram of how a web-of-trust can be constructed.



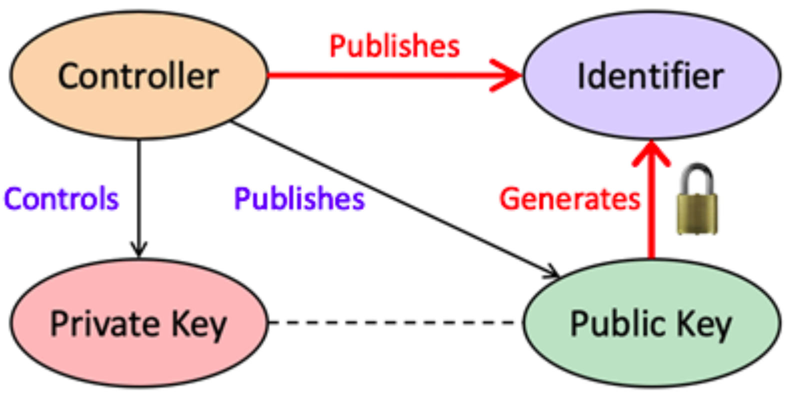
Since the original formulation, hundreds of academic papers have been written exposing issues and proposing improvements for the web-of-trust model[232]. The primary challenge, however, is that the only thing it really changes in the conventional PKI model is ***who*** signs the digital certificate. It turns the problem of "who do you trust" (the TTP problem) into the problem of "who do you trust who knows someone else you can trust", i.e., how do you discover a "trusted path" to the digital certificate you want to verify. So far no one has developed a reasonably secure, scalable, adoptable solution to this problem.

8.3.4 Solution #3: Self-certifying identifiers

The drawbacks of both the conventional and web-of-trust PKI models ultimately led to a very different approach: remove the need for a TTP altogether by replacing it with yet another clever use of cryptography. In other words, rather than trying to reuse an existing identifier for the controller (e.g., domain name, URL, X.500 Distinguished Name) and then binding it to the public key, reverse the whole process and ***generate an identifier for the*** *controller based on the public key itself* (either directly, or via a transaction with a blockchain, distributed ledger, or similar system).

This new approach to constructing the PKI trust triangle can be visualized in figure 8.19.

Figure 8.19: Self-certifying identifiers take a completely different approach to the identifier binding problem by generating the identifier for the controller from the public key



There are two obvious benefits of this approach. First, it solves the public-key-to-identifier binding problem by binding the public key to the identifier the same way the public key is bound to the private key: cryptographically. This makes the identifier **self-certifying** because no external certificate authority or digital certificate is needed to certify the binding with the public key—it can be done cryptographically in milliseconds.

Secondly, it solves the identifier-to-controller binding problem because *only the controller of the private key can prove control of the identifier*. In other words, under the self-certifying identifier model, ***the controller controls all three points of the real PKI trust triangle*** because all three values are generated cryptographically using key material that only the controller possesses.

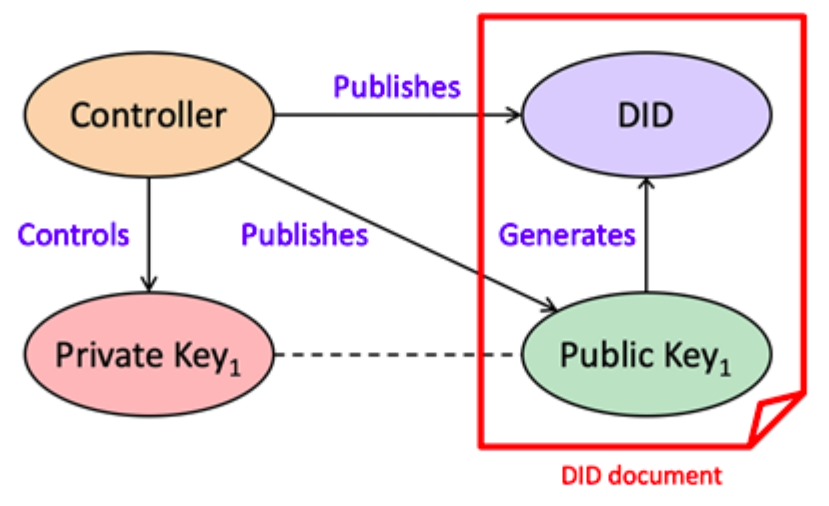
As powerful as this solution appears, completely self-certifying identifiers have one major Achilles heel: the controller’s identifier needs to change every time the public key is rotated. As we will explain further in the chapter devoted to key management, key rotation—switching from one public/private key pair to a different one—is a fundamental security best practice in all types of PKI. Thus the inability for basic self-certifying identifiers to support key rotation has effectively prevented their adoption as an alternative to conventional PKI.

8.3.5 Solution #4: DIDs

What if there was a way to generate a self-certifying identifier once, and then be able to continue to verify it after every key rotation? Enter the DID.

First, the controller generates the original self-certifying identifier—the DID—once, based on the genesis public/private key pair as shown in figure 8.19. Next the controller publishes the original DID document containing the DID and the public key as shown in figure 8.20.

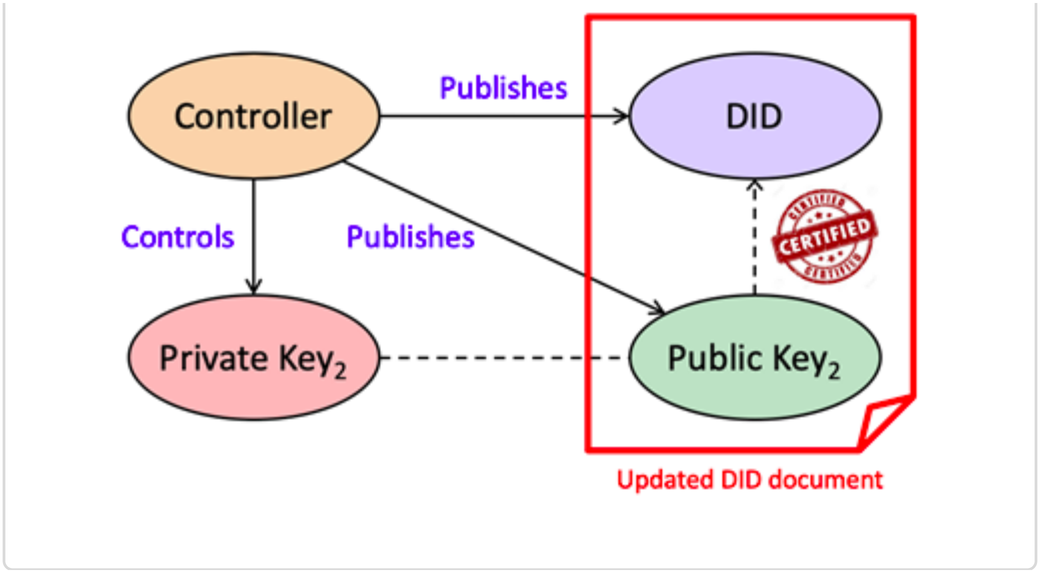
Figure 8.20: The controller publishes the original DID and public key is the original DID document



At this point anyone with access to the DID document can cryptographically verify the binding between the DID and the associated public key—no TTP or other external authority is needed.

Now, when the controller needs to rotate the key pair, the controller creates an *updated DID document* and signs it with the *previous private key* as shown in figure 8.21. Again, this is *self-certification* because it is performed only by the controller—no external authority is needed.

Figure 8.21: The controller publishes an updated DID document containing the original DID and the new public key and then digitally signs it with the original private key, creating a chain-of-trust between the DID documents



This “chain of trust” between DID documents can be traced back through any number of updates to the original DID document with the original self-certifying identifier. Essentially each DID document serves as a new digital certificate for the new public key as shown—but without the need for a CA or any other TTP to certify it.

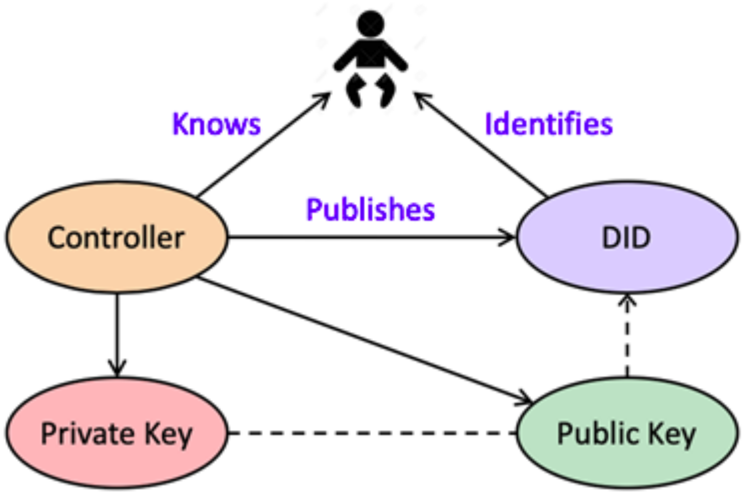
8.4 Four benefits of DIDs that go beyond PKI

So DIDs allow us to finally achieve broad adoption of self-certifying identifiers and still enjoy key rotation and other essential features of conventional PKI—without the drawbacks. But the benefits of DIDs do not stop there. In this section we’ll cover four benefits of DIDs that go ***beyond*** what is offered by PKI as we know it today.

8.4.1 Beyond PKI benefit #1: Guardianship and controllership

To begin with, DIDs provide a clean way to identify entities *other* than the controller of the DID. Conventional PKI generally assumes the registrant of the digital certificate (the controller of the private key) is the party identified by the digital certificate. However there are many situations where this is not the case. Take a newborn baby. If he/she were to need a DID—for example, to be the subject of a birth certificate issued as a verifiable credential—the newborn is in no position to have his/her own digital wallet. He/she would need one of its parents (or another guardian) to issue this DID on its behalf. In this case the entity identified by the DID—the DID subject—is explicitly *not* the controller as shown in figure 8.22.

Figure 8.22: An example of a DID used to identify a DID subject (a newborn baby) that is not **the controller of the DID.**



Of course, newborns are just one case where a DID subject cannot be its own controller. There are dozens more just among humans: elderly parents, dementia patients, refugees, homeless, anyone without digital access. All of these need the concept of

digital guardianship[233]—a third party who accepts the legal and social responsibility to manage a digital wallet on behalf of a DID subject, called the **dependent**. SSI digital guardianship is a very broad, deep, and rich subject that is explored separately in the chapter on governance frameworks.

Digital guardianship only applies to humans, however. What about all the non-human entities in the world? The vast majority are not in a position to issue DIDs either. For example:

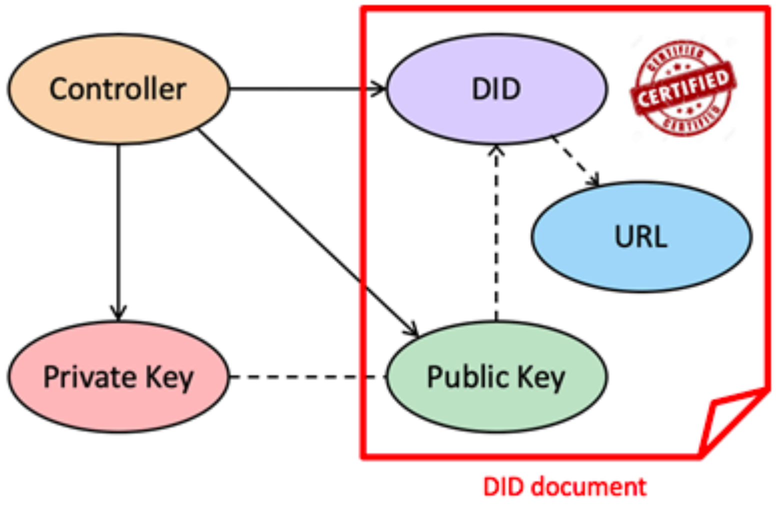
1. **Organizations of all kinds.** Every legal entity in the world that's not an individual human needs some form of identifier in order to operate within the law. Today they have business registration numbers, tax ID numbers, domain names, and URLs. Tomorrow they will have DIDs.
2. **Man-made things.** Virtually everything on the Internet of Things (IoT) can benefit from having one or more DIDs, but relatively few connected things (e.g., smart cars, smart drones) will be smart enough to have their own digital agents and wallets generating their own DIDs—and even then they will still be controlled by humans.
3. **Natural "things".** Animals, pets, livestock, rivers, lakes, geological formations—not only do these have identities but in many jurisdictions they also have at least a limited set of legal rights. So they too can benefit from DIDs.

All of these represent categories of entities that need *third-party controllers*—a relationship that has been called **controllership**[234] to differentiate it from guardianship of a human being. With guardianship and controllership, we can now extend the benefits of PKI to literally every entity that can be identified.

8.4.2 Beyond PKI benefit #2: Service endpoint discovery

The second added benefit of DIDs is their capacity to enable **discovery**, i.e., a way to determine how to interact with a DID subject. To do this, the DID controller publishes one or more service endpoint URLs in a DID document (see figure 8.10 for an example). This three-way binding is shown in figure 8.23.

Figure 8.23: The three-way binding between DIDs, public keys, and service endpoint URLs.



While this makes DID documents generally useful for many types of discovery, it is essential for discovering the **agent endpoints** necessary to remotely establish DID-to-DID connections (below) and communicate via the DIDComm protocol (Chapter 5). In fact one could say that DID-based discovery of agent endpoint URLs is as essential to SSI as DNS-based discovery of IP addresses is to the Web.

8.4.3 Beyond PKI benefit #3: DID-to-DID connections

What made SSL/TLS the largest PKI in the world is that the public key in an X.500 digital certificate can be used for a secure HTTPS connection between a web server and a browser. In other words, it was demand for secure e-commerce, e-banking, e-health and other online transactions that drove the growth of the SSL/TLS PKI.

The same will be true in spades for SSI. Because DID documents can include both public keys and service endpoint URLs, every DID represents the opportunity for its controller to create an instant, secure, private, peer-to-peer connection with any other DID controller. Even better, unlike static public key certificates that must be obtained in advance from a CA, DIDs can be generated immediately, locally, on-the-fly as they are needed for new connections. In fact one DID method has been created exclusively for this purpose: the **Peer DID method**[235].

Developed by Daniel Hardman and the Hyperledger Aries community, peer DIDs do not need a blockchain, distributed ledger, or any other external database. They are generated locally in the DID controller's digital wallet and exchanged directly, peer-to-peer, using a protocol based on Diffie-Hellman key exchange.[236] This creates a connection between the two parties that is not like any other connection in traditional network architecture. Table 8.7 highlights the five special properties of DID-to-DID connections.

Table 8.7: The five properties of DID-to-DID connections

Property	Description
Permanent	The connection will never break unless one or both parties want it to.
Private	All communications over the connection can be automatically encrypted and digitally signed by the private keys for the DIDs.
End-to-end	The connection has no intermediaries—it is secure from “DID to DID”.
Trusted	The connection supports verifiable credential exchange to establish higher trust in any relevant context to any level of assurance.
Extensible	The connection can be used for any application that needs secure, private, reliable digital communications via the DIDComm protocol or any other protocol supported by both agents.

DID-to-DID connections, whether between peer DIDs (the default) or public DIDs, are the centerpiece of Layer Two of the Trust over IP stack described in Chapter 5. Agents at this layer communicate using the secure DIDComm protocol in much the same way web browsers and web servers communicate over the secure HTTPS protocol. DIDs “democratize” the SSL/TLS PKI plumbing so now secure connections can be available to anyone, anytime, anywhere.

8.4.4 Beyond PKI benefit #4: Privacy by Design

At this point it should be obvious how much DIDs can help increase security on the Internet. But while security is necessary for privacy, it is not sufficient. Privacy is more than just preventing private information from being snooped or stolen. It is also ensuring that parties with whom you *do* choose to share your private information (doctors, lawyers, teachers, governments, companies that sell you products and services) protect that information and do not use or sell it without your permission.

So how do DIDs help with *that*?

The answer may surprise you. Conventional identifiers for people are assigned one time by third parties: government ID numbers, health ID numbers, driving license numbers, mobile phone numbers. When you share these as part of your personal data, they make it easy for you to be tracked and correlated across many different relying parties. They also make it easy for those parties to share information and compile "digital dossiers" about you.

DIDs turn that whole equation upside down. Rather than you sharing the same ID number with many different relying parties, you generate and share your own **pairwise peer DID** each time you form a relationship with a new relying party. That relying party will be the only one in the world that knows that DID (and its public key and service endpoint URL). And that relying party will do the same for you.

So rather than you having a single DID, similar to a government-issued ID number, you will have *thousands of DIDs*—one per relationship. Each pairwise-unique peer DID gives you and your relying party your own **permanent private channel** connecting the two of you as envisioned by Philip Zimmermann with PGP. The first advantage of this channel is that you can authenticate each other—in both directions—automatically just by exchanging messages signed by each of your private keys. Spoofing or phishing a relying party with whom you already have a connection will become nearly impossible.

The second major advantage is that peer DIDs and private channels give you one simple, standard, verifiable way to share **signed personal data**—personal data for which you have granted specific permissions data to the relying party. On your side, the benefit is convenience and control—at a glance you can see what you shared with whom and why. On the relying party side, the benefit is fresh, first-person data with cryptographically- verifiable, GDPR-auditable consent—plus an easy, secure way to support all the other GDPR-mandated personal data rights (access, rectification, erasure, objection).[237]

The third major benefit is that, with signed data, we can finally protect both individuals and relying parties from the damage caused by the massive data breaches we read about almost daily (Target, Equifax, Sony, Yahoo, Capital One). What motivates criminals to break into those data silos is the value of that personal data—primarily because the criminals can use it can break accounts all over the Internet.

As those accounts convert over to using pairwise peer DIDs and signed personal data, the value of that personal data *to anyone else but the relying party who has explicit signed permission* disappears. In fact, if you cannot cryptographically prove you have permission to use the data, then not only does the data become worthless—it becomes ***toxic***. Mere possession of unsigned personal data could become a crime. Like toxic waste, it will be something companies, organizations, and even governments will want to get rid of as quickly as possible.

So now you see why many in the SSI community consider DIDs, verifiable credentials, and the Trust over IP stack to be a sea change in privacy on the Internet. While we are still in the very early stages of implementing this new approach, it finally gives individuals new tools for controlling the use of their personal data. And this control, when bundled with the rest of the tools in the Trust over IP stack for building and maintaining digital trust, might actually be a path to putting the privacy genie back in the bottle—a feat that many people believed was impossible.

8.5 The Semantic Level: What DIDs Mean

8.5.1 The meaning of an address

Addresses do not exist on their own. They only exist in the context of a network that uses them. Whenever we have a new type of address, it is because we have a ***new type of network*** that needs this new address to do something that could not be done before. Table 8.7 shows this progression over the past few hundred years.

Table 8.7: An historical perspective on the evolution of addresses for different types of networks

Origin	Address Type	Network

Pre-history	Human name	Human networks (family, clan, tribe, etc.)
~1750	Postal address	Postal mail network
1879	Telephone number	Telephone network
1950	Credit card number	Payment network
1964	Fax number	Fax (facsimile) network
1971	Email address	Email network
1974	IP address	Internet (machine-friendly)
1983	Domain name	Internet (human-friendly)
1994	Persistent address (URN)	World Wide Web (machine-friendly)
1994	Web address (URL)	World Wide Web (human-friendly)
2003	Social network address	Social network
2009	Blockchain address	Blockchain or distributed ledger network
2016	DID	DID network (aka trust network)

So the real meaning of a DID comes down to what can be done with it on a **DID network**.

8.5.2 DID networks and trust networks

Just as everything on the Internet has an IP address—and everything on the Web has a URL—everything on a DID network has a DID. But that begs the question: *why* does everything on a DID network need a DID? What new communications network functionality do DIDs enable that could not be done before?

The short answer is that DIDs were invented to support both the ***cryptographic trust*** and the ***human trust*** required for the four-layer architecture of any **trust network** based on the **Trust over IP stack** introduced in Chapter 5 and shown again here.

Figure 8.24: DIDs are foundational to all four levels of the Trust over IP stack

DIDs are essential to each layer of the stack as follows:

1. **Layer 1: DID Registries.** DIDs published on public blockchains like Bitcoin, Ethereum or distributed ledgers as Sovrin, ION, Element, and Veres One—or on distributed file systems like IPFS—can serve as publicly-verifiable trust roots for participants at all higher layers. They literally form the foundation of a trust layer for the Internet.
2. **Layer 2: DIDComm.** By definition DIDComm is a P2P protocol between agents identified by DIDs. By default

these are pairwise pseudonymous peer DIDs issued and exchanged following the Peer DID specification so they exist only at layer 2. However they can also be public DIDs from layer 1.

3. **Layer 3: Credential Exchange.** As covered in the verifiable credentials chapter, DIDs are integral to the process of issuing and verifying digitally-signed verifiable credentials as well as to the discovery of service endpoint URLs for credential exchange protocols.
4. **Layer 4: Governance Frameworks.** As will be covered in that chapter, DIDs are the “anchor points” for discovery and verification of governance authorities (as legal entities) and governance frameworks (as legal documents), as well as for every other participant role defined in a specific governance framework. DIDs also enable verifiable credentials to persistently reference the governance frameworks under which they are issued, and for governance frameworks to reference each other for interoperability.

In short, DIDs are the first widely-available, fully-standardized identifiers designed explicitly for building and maintaining digital trust networks that are protected by cryptography “all the way down”.

8.5.3 Why isn’t a DID human-meaningful?

Many people have asked: if DIDs are the latest and greatest identifier for the cutting edge of communications on the Internet—then why aren’t they more human-friendly? The answer lies in a conundrum called **Zooko’s Triangle**[238] named after Zooko Wilcox-O’Hearn[239] who coined it in 2001. It is a trilemma that states that an identifier system can only achieve at most two of the following three properties:

1. **Human-meaningful:** All of the identifiers are semantic names (low-entropy) from ordinary human language.
2. **Secure:** Identifiers in the system are guaranteed to be unique, i.e. each identifier on is bound to only one specific entity.
3. **Decentralized:** Identifiers can be generated and correctly resolved to the identified entities without the use of a central authority or service.

Figure 8.x: Zooko's Triangle is a trilemma that proposes an addressing system can have at most two of these three properties

Although some believe Zooko's Triangle can actually be solved, most Internet architects will agree it is far easier to achieve two of these three properties than all three. As this chapter makes clear, with DIDs the two properties chosen were **secure** and **decentralization** (the latter being built right into the acronym "DID"). What was given up—due to the crypto- graphic algorithms used to generate DIDs—was any attempt at being human-meaningful.

But the SSI community realized that while DIDs alone could not solve the human- meaningful naming problem, they could in fact anchor a promising new solution. The trick was not to do it at the public DID layer (layer 1 of the ToIP stack)—or at the peer DID layer (layer 2)—but *at the verifiable credentials layer* (layer 3). In other words, a specific class of verifiable credential could assert one or more **verifiable names** for a DID subject. By creating searchable **credential registries** for the names in these credentials, we could collectively build a naming layer that will be semantically richer, fairer, more trusted, and more distributed than the current DNS naming layer.

Figure 8.26: Verifiable credentials for the human-meaningful names of DID subjects can be layered over machine-friendly DIDs the same way human-meaningful DNS names were layered over machine-friendly IP addresses.

This **verifiable naming layer** would no longer need to be arbitrarily divided into top-level domain (TLD) name registries, but could capture the full richness of human name(s) in any language for any kind of DID subject: person, organization, product, concept, and so on. Furthermore, the authenticity of names—for people, for companies, for products—could be attested to in a decentralized way by issuers of all kinds so spoofing or phishing would become an order of magnitude harder than it is in today’s Internet “Wild West”.

8.5.4 What does a DID actually identify?

We saved this question for last because from a semantic standpoint it is the deepest. The easy answer is exactly what it says in the DID specification: “A DID always identifies the DID subject”. And that is completely true, no matter what that subject may be: a person, organization, department, physical object, digital object, concept, software program—anything that has identity.

Where the confusion arises is with the DID document that a DID resolves to. Since a DID document is a real resource on the Web, and since it describes the DID subject, then isn't a DID document a representation of the DID subject? And if that is true, then ***isn't a plain DID also a URL that identifies the DID document***?

This tips us into one of the oldest and longest debates in the Semantic Web—one so deep it has its own Wikipedia page called **HTTPRange-14**[240] (after the issue number it was assigned by the W3C Technical Architecture Board). The essence of the debate is that URIs may be used to identify resources on the Web ("information resources" such as Web pages or image files), or resources that are not on the Web ("non-information" resources such as people, places, or planets). A single URI however ***can only identify exactly one resource***. So a DID by itself cannot at the same time serve as both an abstract identifier (URN) that identifies the DID subject and as a concrete identifier (URL) that identifies a DID document describing the DID subject.

The editors of the DID specification have proposed a solution to this dilemma, which is:

1. The DID by itself—without any other syntax allowed in a DID URL—always identifies the DID subject.
2. The DID by itself ***plus a single slash ("/") character*** (allowed in DID URL syntax as an "empty path") identifies the DID document.

So, for example, if the following DID was assigned to identify the planet Jupiter:

```
did:example:21tDAKCERh95uGgKbJNHYp
```

copy

Then the following DID URL identifies the associated DID document:

```
did:example:21tDAKCERh95uGgKbJNHYp/
```

copy

The plain DID contained within this DID URL ***by itself still identifies the planet Jupiter***. It is the addition of the forward slash ("/") delimiter character that creates the DID URL that identifies the DID document as a separate resource on the Web.

This pattern of assigning different URIs to: a) a real-world entity, and b) a Web resource that describes it is well-known in the Semantic Web community. For example, it is used by the WebID standard[241] (part of the Solid project)[242] as well as by the semantic database project DBPedia. An example from the latter illustrates perfectly: here is the DBPedia URI that identifies the planet Jupiter ***as an abstract resource***:

```
https://dbpedia.org/resource/Jupiter
```

copy

And here is the URL that identifies a Web page describing the planet Jupiter.

```
https://dbpedia.org/page/Jupiter
```

copy

Note that when the first URI (the one that identifies the planet) is dereferenced (e.g., put into a browser), the DBPedia site automatically redirects to the second URL. This way, it is possible to have separate, semantically clean identifiers for the abstract non-information resource and the concrete information resource while at the same time supporting the functionality developers expect—which is that if you start with an abstract identifier (in our case, a plain DID), you end up with a Web resource that describes it (in our case, a DID document).

While to some people this might seem like an academic debate akin to “how many angels can dance on the head of a pin”, in fact this level of semantic precision is very important when it comes to building a strong foundation for trust on the Internet. It underscores that DIDs are the **first digital identifiers designed for universal verifiable identification** of any entity of any kind anywhere.

Although it is one of the longest chapters in this book, if you’ve read it this far, you now have a much deeper understanding of why we call DIDs “the atomic building block of digital trust”. They are far more than just a new type of globally unique identifier. By relying on cryptography for both generation and verification, DIDs change the fundamental power dynamics of the Internet. Instead of scale laws pulling everything into the gravity well of the Internet giants at the center, DIDs pushes power to the very edges—to the individual digital agents and wallets where DIDs are born and DID-to-DID connections are made. This “Copernican inversion” creates the new spacetime of decentralization—a universe where entities of all kinds can be “self-sovereign” and interact as peers using all four layers of the Trust over IP stack.

For our next chapter we’ll move up to Layer Two of the stack to understand the digital wallets and digital agents required to generate DIDs, form DID-to-DID connections, exchange verifiable credentials, and handle decentralized key management.

[213] https://github.com/WebOfTrustInfo/rebooting-the-web-of-trust/blob/master/final-documents/dpki.pdf

[214] Secure Sockets Layer (SSL) and Transport Layer Security (TLS) For more details please access https://en.wikipedia.org/wiki/Transport_Layer_Security

[215] https://www.w3.org/TR/did-core/

[216] https://www.w3.org/2019/did-wg/

[217] “Resource” is the term used by the World Wide Web Consortium (W3C) across all Web standards. The digital identity community generally uses the term “entity”. For the purposes of this book, the two terms can be considered equivalent.

[218] https://tools.ietf.org/html/rfc3986

[219] https://tools.ietf.org/html/rfc8141

[220] See https://en.wikipedia.org/wiki/Universally_unique_identifier#Collisions

[221] In the future, "DID navigators" may be able to let you travel across a "trust web", but most likely this will be done by associated conventional Web pages with DIDs.

[222] https://w3c-ccg.github.io/did-method-registry/

[223] https://www.w3.org/2019/09/did-wg-charter.html

[224] https://en.wikipedia.org/wiki/Domain_Name_System_Security_Extensions

[225] https://en.wikipedia.org/wiki/IDN_homograph_attack

[226] https://en.wikipedia.org/wiki/DNS_spoofing

[227] https://en.wikipedia.org/wiki/Public_key_certificate

[228] https://en.wikipedia.org/wiki/Trusted_third_party

[229] https://www.cpacanada.ca/en/business-and-accounting-resources/audit-and-assurance/overview-of-webtrust-services

[230] https://en.wikipedia.org/wiki/Public_key_infrastructure#Web_of_trust

[231] https://en.wikipedia.org/wiki/Web_of_trust

[232] https://en.wikipedia.org/wiki/Web_of_trust#WOT_assisting_solutions

[233] https://sovrin.org/guardianship/

[234] See the definition of "Controllership" in https://sovrin.org/library/glossary/

[235] https://openssi.github.io/peer-did-method-spec/

[236] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[237] https://en.wikipedia.org/wiki/General_Data_Protection_Regulation#III_Rights_of_the_data_subject

[238] https://en.wikipedia.org/wiki/Zooko%27s_triangle

[239] Zooko worked in the 90s with famous cryptographer David Chaum developing DigiCash and also founded Zcash, a cryptocurrency aimed at using cryptography to provide enhanced privacy for its users. The academic cryptography, SSI and public blockchain space overlap on many levels as you read through his book.

[240] https://en.wikipedia.org/wiki/HTTPRange-14

[241] https://www.w3.org/2005/Incubator/webid/spec/identity/

[242] https://solid.mit.edu/

Up next...
12 How open source software helps you control your self-sovereign identity