Reference – <https://ssimeetup.org/did-resolution-given-did-how-do-retrieve-document-markus-sabadello-webinar-13/>

**DID**

A screenshot of a computer

Description automatically generated

**DID Document**

* Should contain only the minimum amount of machine-readable metadata required to enable trustable interaction with the DID subject.
* Contains:
* Set of public keys

Used to authenticate the DID subject during interaction.

* Set of service endpoints

Used for interaction via protocols by those services. (like social networking protocol, identity agent personal data store protocol, etc )

* Authentication methods
* Timestamps, proofs
* Other identifier metadata
* DID Document is completely public and does not need any authentication or authorisation.
* It is not stored as a plain text but instead dynamically constructed by DID resolver.

A screenshot of a computer

Description automatically generated

The DID document contains a single DID with one authentication mechanism using an Ed25519 verification key and one service, which is a verifiable credential service accessible at "https://example.com/vc/".

Service Section In DID Document

The "service" section allows DIDs to associate different types of services with the identity, enabling the DID subject to advertise and communicate its capabilities, endpoints, or resources. This can be useful in various scenarios, such as when an entity wants to communicate the locations of specific services it provides or expose certain functionalities that can be accessed using the DID.

The "service" section typically contains the following properties: id: A URI that uniquely identifies the service entry within the DID document. type: A string or an array of strings indicating the type of service being offered. It can be a predefined type, such as "MessagingService" or "PaymentService," or custom types. serviceEndpoint: The endpoint (URL or other communication channel) where the service is accessible.

{

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

"id": "did:example:123456789abcdefghi",

"service": [

{

"id": "1",

"type": "MessagingService",

"serviceEndpoint": "https://example.com/messages"

},

{

"id": "2",

"type": "PaymentService",

"serviceEndpoint": "https://example.com/payments",

"description": "Payment service for processing transactions."

}

]

}

In this example, the DID subject is associated with two services: a "MessagingService" accessible at "https://example.com/messages" and a "PaymentService" accessible at "https://example.com/payments."

**DID Resolution**

* Depends on the DID methods, there are some DID methods that doesn’t require any network interaction at all and some that requires communication with the blockchain.

**Use Cases of DID Resolution**

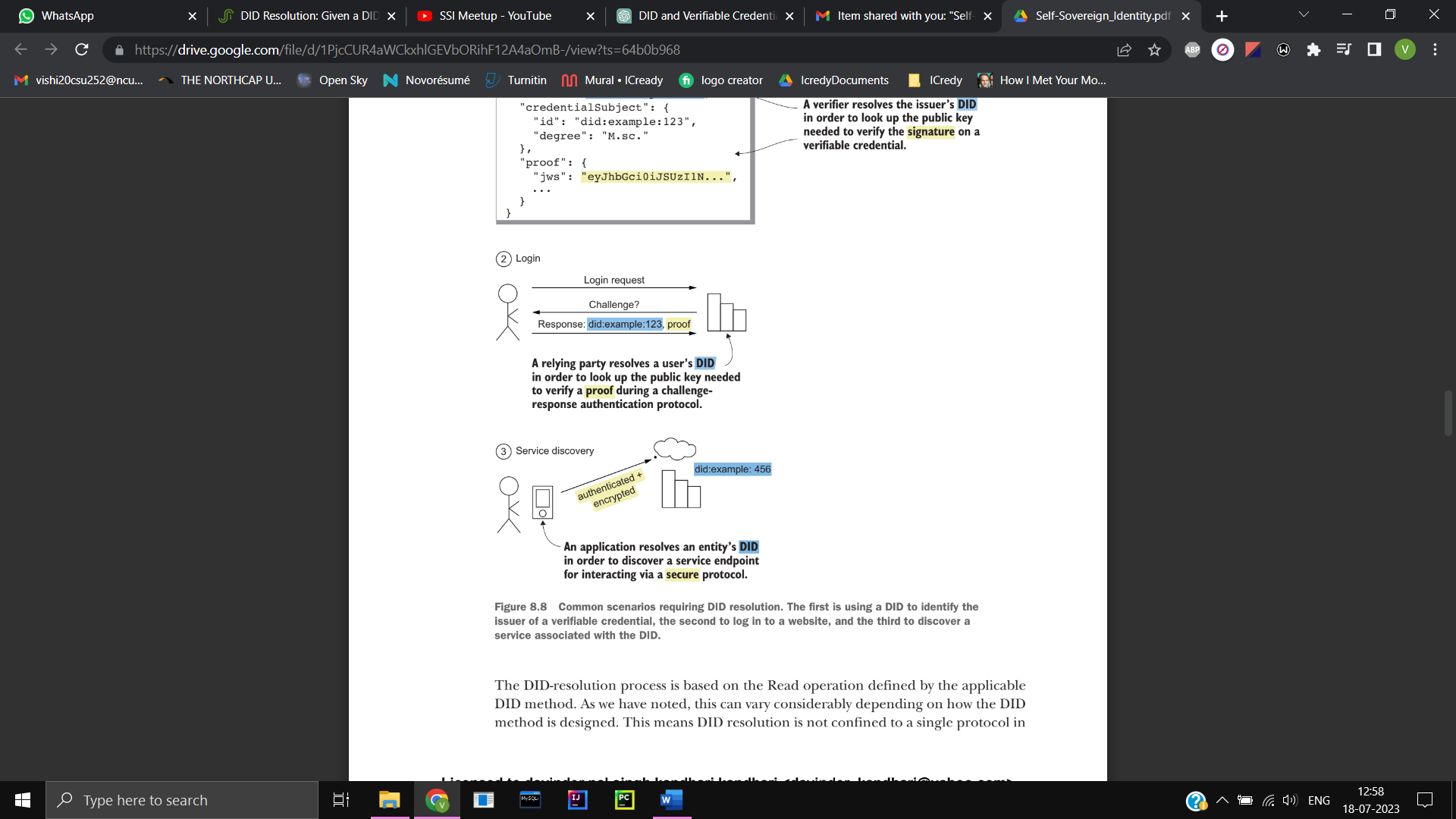
* To look up a public key to verify a digital signature from the issuer of a verifiable credential.

A screenshot of a computer

Description automatically generated

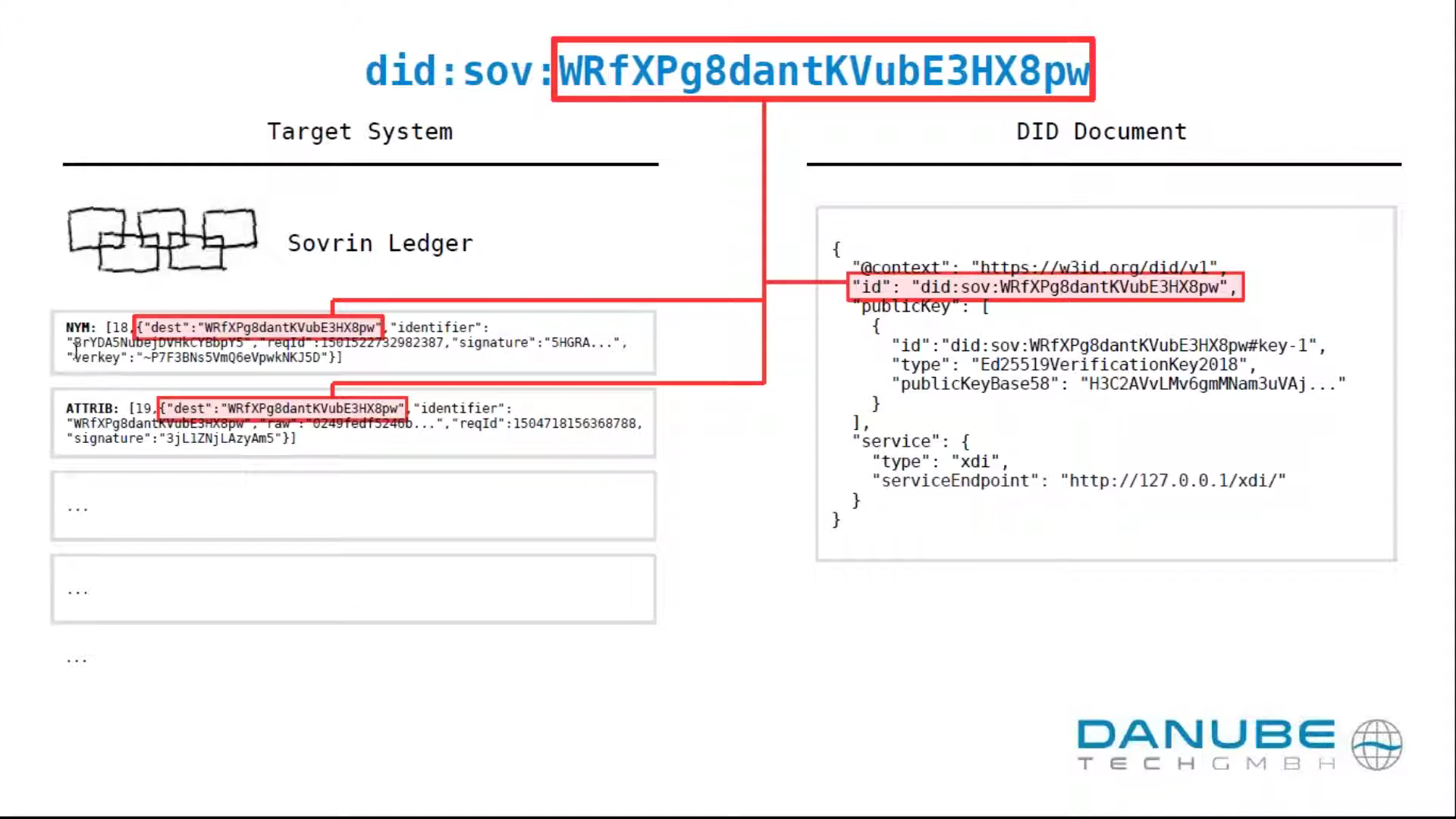
* To discover and access a well-known service associated with the DID controller,

such as a website, social network, or licensing authority



**DID Resolution for Sovrin Method**

* Sovrin natively supports DIDs, but it does not have built-in support for DID documents.
* Starting with the method-identifier, we search for transactions on the Sovrin Ledger.
* In this case, there are two transactions: the nym transaction, used to establish the DID, and the attribute transaction, used to add additional information to the DID.
* The resolver stores the identifier found in the transaction as the "id" in the DID Document.

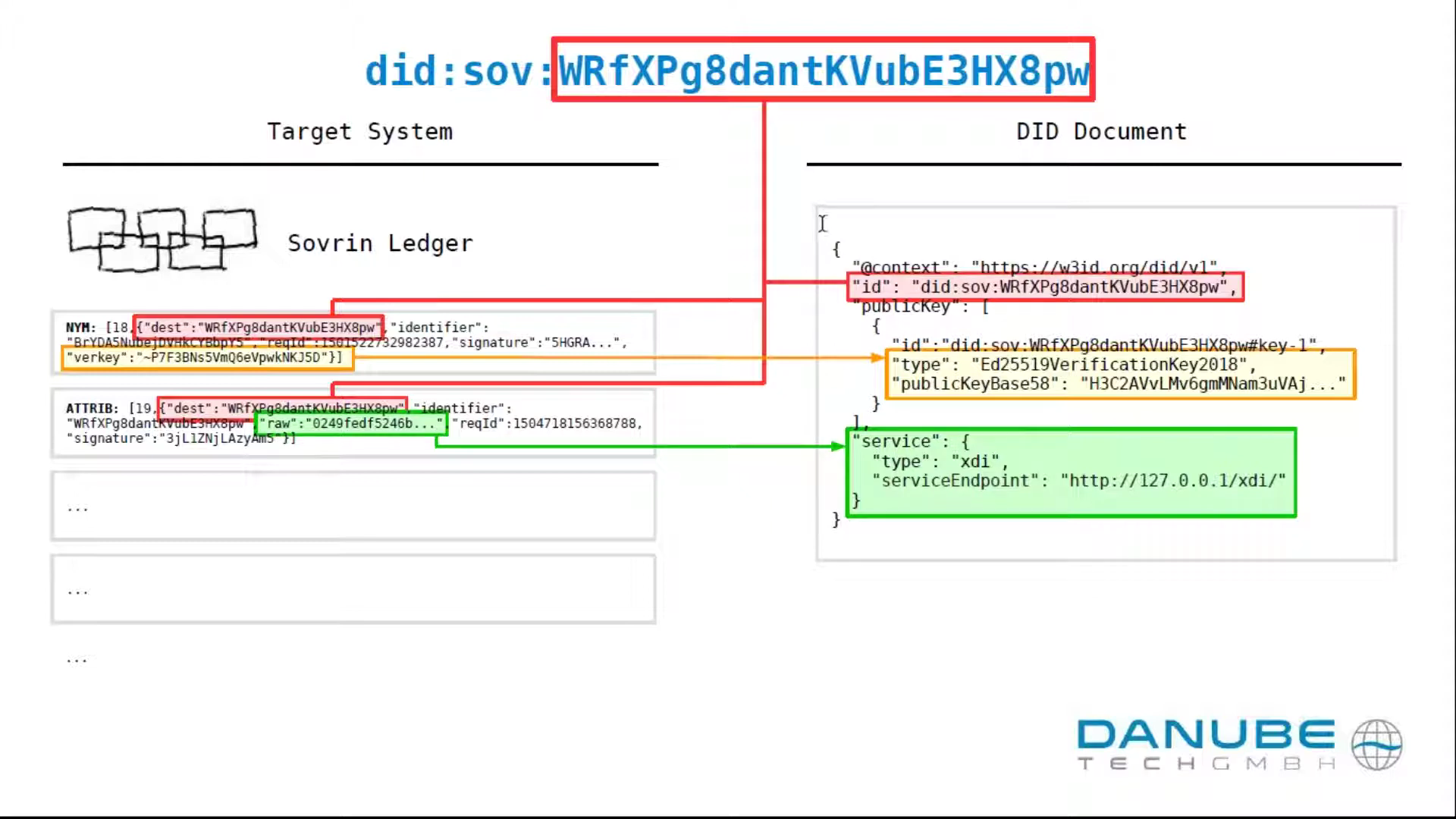


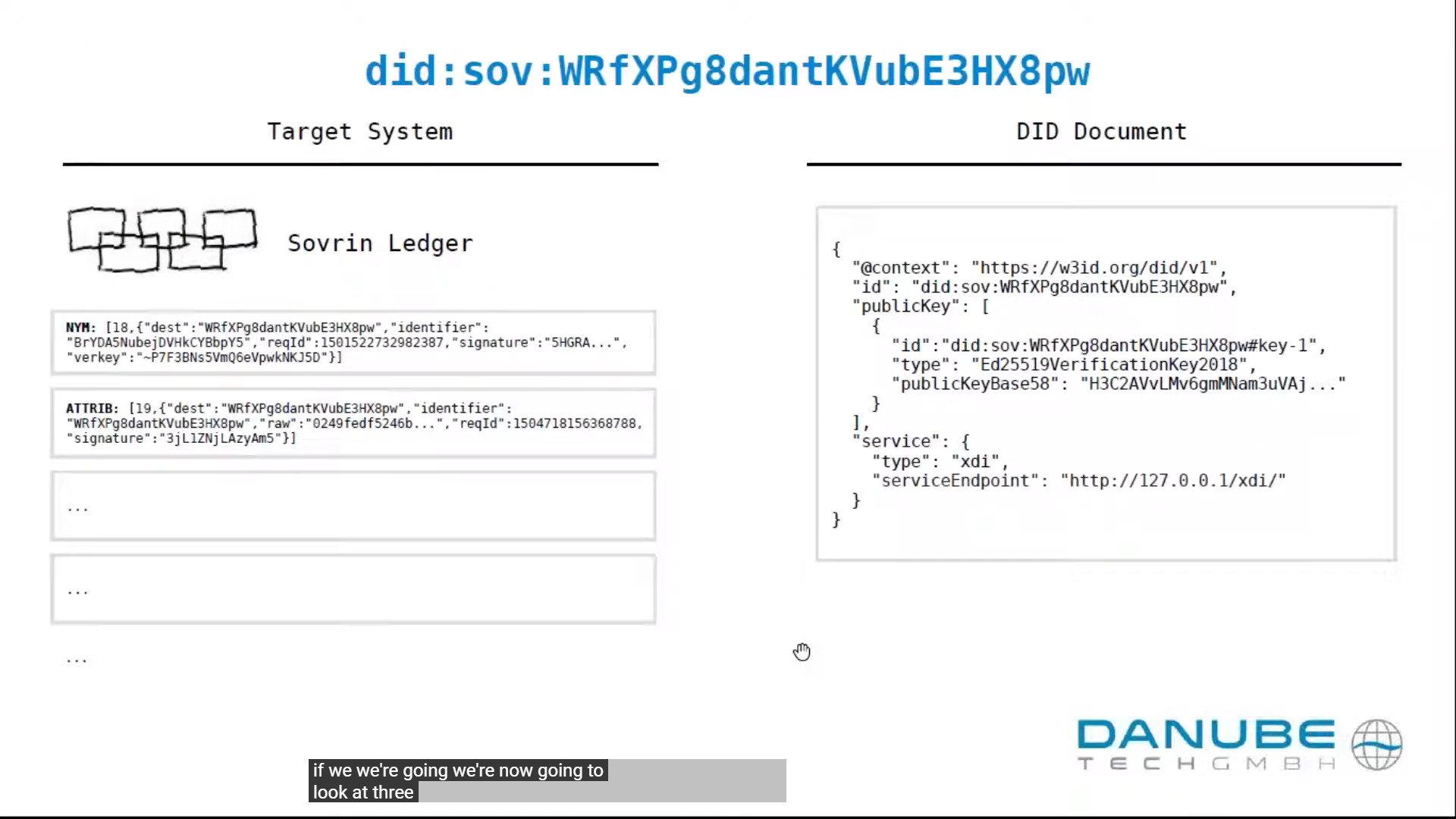
* The public key is derived from the verkey in the nym transaction.
* The resolver will map the verkey to an entry in public key array in DID Document

A screenshot of a computer

Description automatically generated

* The attribute transaction contains extra information encoded in a specific format, which represents an endpoint.
* The resolver will extract and decode this information, and dynamically generate a service entry in the DID Document.





**DID Resolution for Key Method**

* Start with an empty object called "document." This will be the resulting DID Document.
* Split the given did:key identifier into its components: scheme, method, version, and multibaseValue. If there are only three components, assume the version is 1 and use the last component as the multibaseValue.
* Check if the input identifier is valid. The scheme must be "did," the method must be "key," the version must be a positive integer, and the multibaseValue must be a string starting with the letter "z." If any of these requirements are not met, raise an error indicating an invalid DID.
* Create a signatureVerificationMethod by applying the § 3.1.2 Signature Method Creation Algorithm to the identifier, multibaseValue, and options.
* Set the "id" property in the document to the given identifier. If the resulting value is not a valid DID, raise an error indicating an invalid DID.
* Initialize the "verificationMethod" property in the document as an array and add the signatureVerificationMethod as its first item.
* Initialize the "authentication," "assertionMethod," "capabilityInvocation," and "capabilityDelegation" properties in the document as arrays, and set their first items to the "id" value from the signatureVerificationMethod.
* If the option "enableEncryptionKeyDerivation" is set to true, create an encryptionVerificationMethod by applying the § 3.1.5 Encryption Method Creation Algorithm to the identifier, multibaseValue, and options.
* Add the encryptionVerificationMethod value to the verificationMethod array.
* Initialize the "keyAgreement" property in the document as an array and set its first item to the "id" value from the encryptionVerificationMethod.
* Initialize the "@context" property in the document by applying the § 3.1.7 Context Creation Algorithm to the document and options.
* Return the resulting document.

In summary, this algorithm takes a did:key identifier, validates it, creates signature and encryption verification methods, and constructs a DID Document based on the given options. The resulting document includes various properties such as the identifier, verification methods, authentication methods, and context.

**Resolver Metadata**

* Which driver was used
* Duration of resolution process
* Versioning informing about DID Document

**Method Metadata**

* Sovrin: State proofs from ledger

**Universal Resolver**

The retrieval of DID documents and resolution of DIDs are handled by the drivers. Each driver is designed to handle a specific method, enabling the resolution process. In cases where certain methods inherently support DID documents, dedicated complex drivers are not required as the resolver can directly download the associated DID document.

The interaction between the resolver, its driver, and the target system varies depending on the specific method employed.

A screenshot of a computer screen

Description automatically generated

Ideally, it is preferable for the driver to be closely integrated with the target system. For instance, when dealing with Sovrin DIDs, having local Sovrin observer nodes or validator nodes would be ideal. However, practically speaking, it is not always feasible to run one's own Sovrin observer node. In such cases, Sovrin offers a feature called state proofs, enabling the receipt of ledger results through intermediaries.