Ethereum Decentralized Digital Identity Trust Services

Antoine DETANTE

Fabrice CROISEAUX

InTech

EDDITS Whitepaper

Abstract

Digital identities and authentication services exist now for more than 15 years and are commonly used either by internet users and business users. Even if they share the same concepts (identity, authorisations, …) these two categories of users have different needs and use today different tools :

- **Internet users** need global and easy access to their identity. They don't want to install dedicated software on their computer or mobile and ideally, they want the management of their identity as transparent as possible. For these reasons, Facebook, Twitter or Google identity are often used to share identity information with external applications.
- **Business users** need trust and legal proof of actions performed with digital identity. The recent eIDAS European Directive describes the characteristics that digital IDs and authentication services must comply with to be acceptable by a court. Even if the regulation has been created in 1999 (The Electronic Signatures Directive 1999/93/EC), compliant identities and electronic signatures are not widely used today in business relationships. They all rely on PKI Infrastructure and X509 Certificates. Strong authentication and qualified signatures (the one that are recognized by a court) need complex installations and lack of interoperability. Even if the eIDAS directive never refers to PKI and certificates, there is today no solution that is eIDAS compliant and that doesn't use PKI Infrastructure.

Appeared with bitcoin in 2009, the blockchain technology is a new category of software platform that gives ease of access and strong trust without any centralisation. It is not the purpose of this document to details what blockchain is and how it works. There is tons of well understandable articles on internet that explain blockchain to readers who are not familiar with this technology. ***Our goal is to leverage the capabilities offered by the blockchain technology to overcome constraints of current strong authentication and signature services***.

Why Blockchain ?

Strong trust

As of today, the main public blockchain's protocols (Bitcoin and Ethereum) haven't been cracked. It is now commonly recognized that a transaction in these public blockchains cannot be modified or deleted by anybody and that blockchain's accesses are as secured as the security of the private key storage. That means that it is not possible to steal any assets or identity owned on the blockchain and also not possible to create faked assets or identity.

General availability and accessibility

As their name may suggest, public blockchains are accessible anytime and anywhere by anybody. It takes only few minutes for any internet user to create a public address on Ethereum and to exchange value or execute transaction on it. It is far more simple than installing complex strong authentication PKI based software.

Ease of deployment

Since the infrastructure is accessible by everybody, deploying or using a new service on the blockchain is a matter of minutes. You don't need to deal with complex infrastrusture installation mechanisme. Public blockchains are by nature 24x7 available.

ERC-725 and ERC-735

KYC on public blockchain is mandatory for any project that needs governance or regulation. ICO's definitely need strong KYC and any project that wants to protect its users and investors will need a strong KYC. It is not possible to perform such a KYC without a commonly shared identity model. That's why ERC-725 and ERC-735 have been created. The goal of these standards is to define a commonly accepted identity and claim model. ERC-725 identity can hold keys to sign actions (transactions, documents, logins, access, etc), and claims, which are attested from third parties (issuers) and self attested, as well as a proxy function to act directly on the blockchain.

EDDITS Whitepaper

The project

We use the public Ethereum blockchain to provide private and business users easy to use strong authentication and signature decentralized services. We want these services to be compliant with the EIDAS Regulation. A legal assessment will be performed by a law firm. We plan to deploy the platform in three steps.

- Ethereum PKI Certifier
- Decentralized Authentication
- Decentralized Signature

ETHEREUM PKI CERTIFIER

Since the Byzantium hard fork and the inclusion of EIP 198, it is possible to verify RSA Signature directly on the EVM with low gas fee. This allows the owner of a X509 Certificate to link its certificate to its blockchain address. We can prove in a trustless way that the same person owns both the private key linked to the X509 Certificate and the private key linked to an Ethereum address.

To do that, our DApp signs the user certificate with the certificate itself and passes both the signature and the certificate as parameters to a call to a Smart Contract. This unambiguously links the user's personal information to the relevant blockchain asset. The PKI Certifier Smart Contract maintains a list of all certified blockchain addresses with the related certificate that contains the full name and the certificate serial number.

DECENTRALIZED AUTHENTICATION

EDAS is an authentication framework. It works by delegating the user authentication to a decentralized web application, using HTTP redirect binding (in the SAML way). Users use their Ethereum keys to prove access to an ERC-725 "Identity" Smart Contract.

The authentication flow is designed to be cost-free: no Ethereum transaction occurs during the authentication.

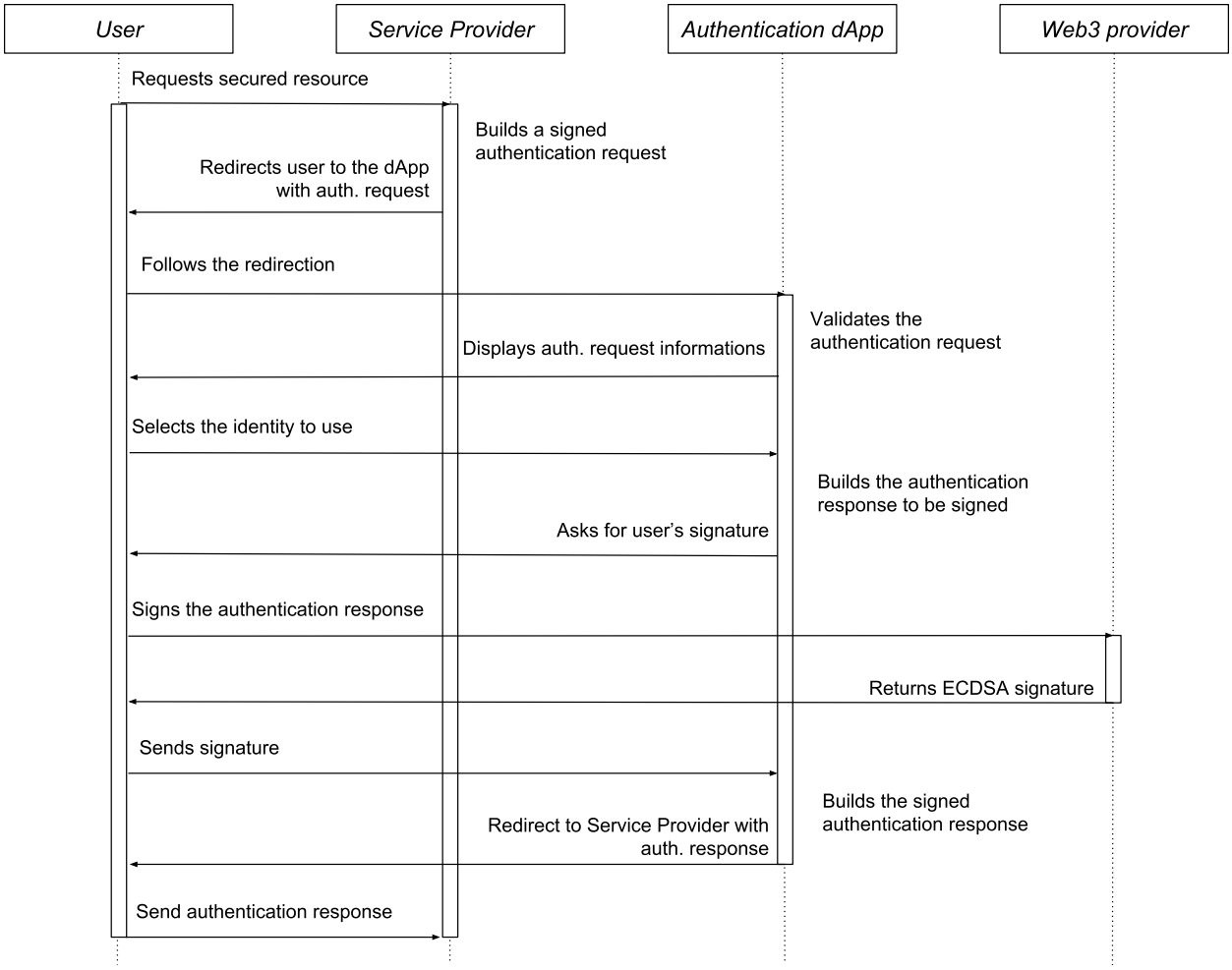
Roles

User wants to be authenticated on a *Service Provider*. User has access to one or more Ethereum private keys associated to a ERC-725 Identity.

Service Provider ("SP") authenticates users to give access to web resources. *Service Provider* has access to an Ethereum private key, eventually linked to an ERC-725 Identity, to prove its identity to the *User* during the authentication flow.

Authentication dApp is a client-side JavaScript application (running in the *User* browser), used during the authentication flow to validate the *Service Provider* authentication request and produce an authentication response signed with one of the *User* keys.

Authentication flow



ECDSA signature

Two signature operations take place during the authentication flow:

- The *Authentication Request* is signed by the Service Provider, to prove its identity to the User.
- The *Authentication Response* is signed by the User, to prove he has access to a key linked to the ERC-725 Identity Smart Contract.

Although the process is the same for the two signatures, the context is different. SP signature occurs server side: the private key can be securely stored (hard drive, dedicated security hardware, ...). On the over hand, the User signature operation occurs in the browser: user's private key must be available in the browser's sandbox. To secure the signing operation for the user, the Authentication dApp uses the `personal_sign` Ethereum JSON-RPC call (using the Web3 provider injected by MetaMask).

`personal_sign` adds a special prefix to the message before applying the hash function. For consistency, the two signature operations will be calculated in the same way:

```
sign(keccack256("\x19Ethereum Signed Message:\n" + len(m) + m))
```

where *m* is the message to be signed and *sign* the ECDSA signature based on the secp256k1 curve.

Authentication Request and Response format

Authentication requests and responses are signed tokens passed in the URL between the authentication dApp and the SP. To be consistent with common practice, these signed tokens are modeled as JSON Web Tokens (JWT) using a custom signature scheme (due to specific ECDSA signature described previously).

The general format of these JWT is `header.payload.signature` where:

- The **header** is defined as:
  `{ "typ": "JWT", "alg": "ESK256" }` encoded with `base64url`
- The **payload** is defined as:
  a JSON object (specific to Auth. Request / Response) encoded with `base64url`
- The **signature** is defined as:
  a ECDSA signature with an Ethereum private keys of the following hash:

```
keccack256(
  "\x19Ethereum Signed Message:\n" +
  len(header + "." + payload) +
  header + "." + payload
)
```

  encoded with `base64url`

Note that by default `personal_sign` returns the signature in an hexadecimal format: the signature must be hex-decoded before being encoded with `base64url`.

Authentication Request JWT payload

The payload built by the SP has the following shape:

```
{
    "sub": "0x32be343b94...88",
    "name": "My Service Provider",
    "redirect": "https:/www.my-service-provider.lu/login"
}
```

where:

- `sub` identify the Service Provider. It must be the address linked to the private key which signs the JWT, or the address of a ERC-725 Identity Smart Contract. If `sub` is an ERC-725 address, the key which signs the JWT must be linked as ACTION key on this identity.
- `name` is the label of the Service Provider. It will be displayed on the dApp login screen as "Please confirm the connexion to *name*"
- `redirect` is the URL where the user will be redirected after confirmation of authentication. The redirection is performed through a HTTP GET request, adding the authentication response in the URL like: `https://www.my-service-provider.lu/login#response=header.payload.signature`

Service Provider can include a `nonce` attribute in the Authentication Request payload with any alphanumeric string (to mitigate replay attacks). If `nonce` is present in the request, the value will be copied in the response built by the dApp.

Authentication Response JWT payload

The payload built by the dApp has the following shape:

```
{
    "sub": "0x22beee3b94...96"
}
```

where:

- `sub` is the address of the ERC-725 Smart Contract selected by the user for the authentication. The Ethereum keys used to sign the JWT must be linked in this ERC-725 contract as a ACTION key.
- If `nonce` was present in the Authentication Request, the payload of the response must include the same value in a `nonce` attribute.

Validation of the Authentication Request

When processing the Authentication Request, the Authentication dApp must performs the following controls:

- Validate the JWT signature and retrieve the (Ethereum) address of the signer
- If the signer's address is equals to the `sub` attribute of the payload, token is VALID
- If the signer's address is not the `sub`, then `sub` must be a ERC-725 address. dApp must read the contract storage to check that the signer's key is defined as ACTION key in the identity. If the key is linked to the identity, token is VALID

Validation of the Authentication Response

When processing the Authentication Response, the Authentication dApp must performs the following controls:

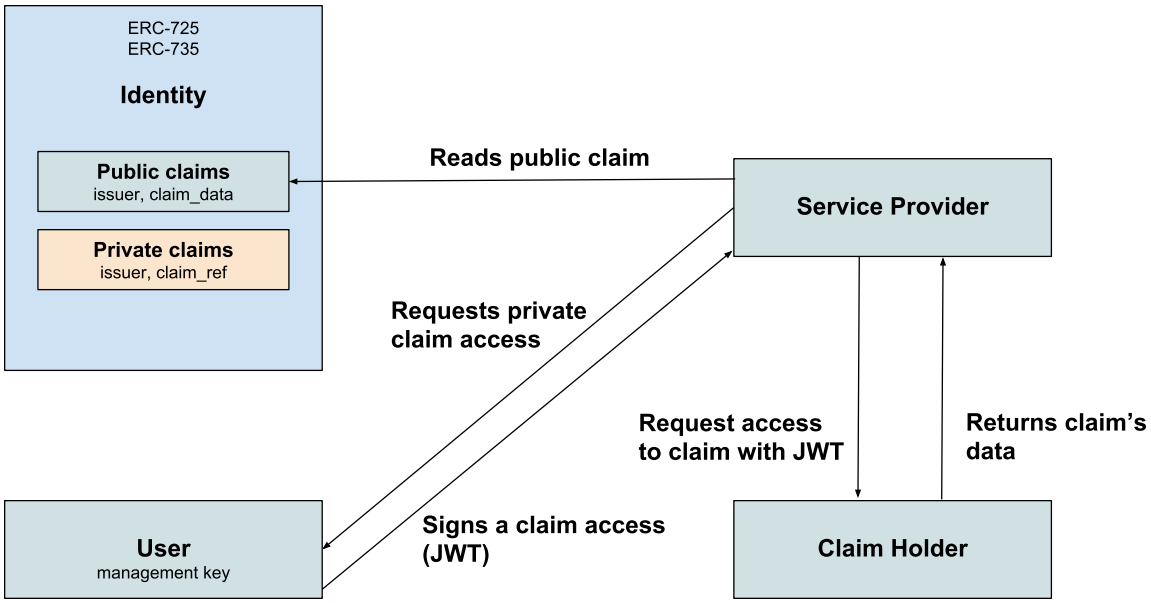
- Validate the JWT signature and retrieve the (Ethereum) address of the signer
- Call the ERC-725 Identity contract at the `sub` address in the JWT payload
- The JWT signer's key must be referenced in the ERC-725 contract as an ACTION key

Claims management

Claims can be collected by the Identity holder to provide proof for attributes linked to his identity. Attributes can be names, postal address, email addres, … and can be issued by any entities. As described in ERC-735, claims are signed by the Claim Issuer.

Claims have a "scheme" which describe the process to validate the claim. By using custom schemes, Claims can be public (ie publicly accessible by blockchain clients or Smart

Contract) or private (ie stored on an external storage). For private claims, EDAS will propose a way, based on signed JWT issued by the identity holder, to retrieve the attribute value on the external storage.



DECENTRALIZED SIGNATURE SERVICE

This service will be developed in a later stage. We will update this document accordingly.

References

Bythantium Hard Fork Announcement and Description. October 12th 2017. https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/.

X509 Certificate Description. https://en.wikipedia.org/wiki/X.509

EIP-725 Identity : https://github.com/ethereum/EIPs/issues/725

EIP-735 Claim Holder : https://github.com/ethereum/EIPs/issues/735