Copyright Notice

This is a preprint of the following chapter:

L. Perugini, A. Pino, A. Solavagione, D. Margaria, and A. Vesco, Integrating the Self-Sovereign Identity in the TCP/IP Stack While Preserving Interoperability with Existing Identity Models, published in: Networking Data Integrity and Manipulation in Cyber-Physical and Communication Systems, edited by: R. Herrero, 2025, Springer, Cham, part of the book series: Engineering Cyber-Physical Systems and Critical Infrastructures (ECPSCI), vol. 13, reproduced with permission of Springer Nature Switzerland AG.

The final authenticated version is available online at:
https://dx.doi.org/10.1007/978-3-031-83149-2_5

Cite as:

L. Perugini, A. Pino, A. Solavagione, D. Margaria, and A. Vesco, Integrating the Self-Sovereign Identity in the TCP/IP Stack While Preserving Interoperability with Existing Identity Models. In: Networking Data Integrity and Manipulation in Cyber-Physical and Communication Systems. Engineering Cyber-Physical Systems and Critical Infrastructures. R. Herrero (eds), vol. 13, Cham: Springer, 2025, pp. 95–117. DOI: 10.1007/978-3-031-83149-2_5.

Integrating the Self-Sovereign Identity in the TCP/IP Stack while Preserving Interoperability with Existing Identity Models

Leonardo Perugini[0009−0004−9203−7085]
Alessandro Pino[0009−0002−1695−8902]
Alberto Solavagione[0009−0002−3871−5370]
Davide Margaria[0000−0002−5275−2138]
Andrea Vesco[0000−0001−7431−6655]

Abstract Self-Sovereign Identity (SSI) is an emerging decentralised identity model that gives peers full control over the data they use to create and prove identity. This new model is based on three fundamental elements: Verifiable Data Registry as the root-of-trust for public keys, Decentralised IDentifiers and Verifiable Credentials as the key identity components currently being standardised by the World Wide Web Consortium (W3C). Most of today's research and implementations use SSI at the application layer of the TCP/IP stack. While this approach makes SSI easier to adopt, it limits the overall benefits of using SSI. Considering a generic authentication process, a client establishes a Transport Layer Security (TLS) channel and authenticates the server using the server's certificate. The server then authenticates the client, which sends its Verifiable Credential over TLS at the application layer. While this approach is suitable for Web scenarios, the use of certificates in large-scale Internet of Things (IoT) systems requires human intervention and entails high management costs. This drawback, widely recognised by industry players, led us to explore the use of Verifiable Credential (VC) in the TCP/IP layers below the application layer to avoid the use of certificates and take full advantage of SSI. This chapter provides an overview of the design choices we made for TLS and IKEv2 in IPSec, and finally our efforts to migrate the SSI to post-quantum cryptography.

1 Introduction

The Internet works without an identity layer, and there is no easy way to know who and what a user is connecting to. With the TCP/IP stack [1], all a user knows is the address of the machine it is connecting to, but that tells him nothing about the subject controlling that machine. Public key certificates play an important role in security

L. Perugini, A. Pino, A. Solavagione, D. Margaria, A. Vesco
Cybersecurity Research Group, LINKS Foundation, 10138 Torino, Italy.
e-mail: {name.surname}@linksfoundation.com

because they provide verified assurance that the party users are connecting to can be trusted. Given the fact that a trusted third party issues these certificates, users can be confident that the information received comes from a trusted party.

The actual history of the Internet tells of the use of two main identity models for users. The first is called siloed identity, where a user has a credential (*i.e.* an account) on a system. The username and password identify the user in that specific system, but they are not a true identity and, more importantly, this pseudo-identity is not persistent. The account has no meaning or value on other systems on the Internet. The second is called federated identity, where a user has an account with one identity provider. The user can access multiple systems under the same federated domain using a single set of credentials. This model is designed to simplify access to multiple systems, but again it is not a rich identity that describes the user's identity, and again it is not persistent because its validity is limited to systems that are part of the federation.

An emerging identity model is called Self-Sovereign Identity (SSI) [2]. It is a decentralised digital identity model that gives each user full control over the data they use to create and prove their identity. SSI is a decentralised, descriptive, persistent and peer-to-peer identity model that has the potential to be used for authentication and authorisation across the Internet. Any two peers (as people, things, machines) can interact in a trusted manner without the need for a trusted third party to create and validate their identity.

The SSI model relies on three fundamental elements: Verifiable Data Registry (VDR) as the Root-of-Trust (RoT) for public keys, Decentralized IDentifier (DID), and Verifiable Credential (VC) as the key component of the identity. Both DID [3] and VC [4] are in the process of being standardised by the W3C.

Each SSI-aware peer generates its identity key pair and stores the public key in the VDR for others to authenticate. A peer's DID represents the address in the VDR where other peers can retrieve its public key. Once these two components have been generated, a peer can request a VC from one of the available Issuers. The VC contains the metadata and claims about the identity of the peer holding it. The purpose of a VC is to describe the identity of the peer, just like any other physical credential in our real world. The combination of the key pair, the DID and at least one VC shapes the digital identity in SSI ecosystems. This composition of the digital identity reflects the decentralised nature of the SSI model. A node builds its own identity and, most importantly, is able to update its own identity in a self-sovereign manner. A node can also revoke its own identity. There is no trusted third party that provides all three components of identity, and no trusted third party is able to fully revoke an identity. In addition, a node can enrich its identity with multiple VCs issued by different Issuers.

Most discussions and proposals for SSI-based authentication consider an implementation at the application layer of the TCP/IP stack. For example, it is envisaged that users will authenticate to their web services by exchanging a self-sovereign identity over a secure communication channel established using TLS with an X.509 certificate at the server side. While an application layer approach is perfectly fine in

Web scenarios, it is not the most suitable in the case of connected systems (*e.g.* IoT systems, critical infrastructures).

Connected systems today make use of public key certificates for security purposes. However, it is widely believed that the adoption of a Public Key Infrastructure (PKI) with X.509 certificates [5] is not the most appropriate solution for managing identities in large-scale deployments with numerous IoTs. The main problem is the high cost of managing certificates throughout their lifecycle, from installation to regular updates and revocation. In addition, the centralised nature and excessive trust in the Certificate Authority (CA) makes PKI vulnerable to single points of failure [6]. The process used by a CA to issue and revoke certificates lacks transparency, the compromise of a CA serving a large system requires a full update of all IoT identities before network operations can resume, and there is no standard enrolment solution tailored to IoT platforms and protocols [7, 8, 9, 10, 11, 12].

The SSI model provides an alternative solution to PKI to reduce the complexity and associated costs of X.509 certificate management in large scale IoT systems. However, the full potential and benefits of SSI can only be realised by adopting it while avoiding the use of certificates on the server side. An SSI native IoT node reduces the need for human intervention in identity management and therefore has the potential to significantly reduce the complexity and cost of identity management. Given this advantage, the research question is how to integrate SSI for authentication purposes into lower layers of the TCP/IP stack while maintaining interoperability with existing identity models such as X.509 certificates and raw public keys.

This chapter provides an overview of our approach to answering the research question, focusing on describing the design choices made in the case of Transport Layer Security (TLS) and in Internet Key Exchange (IKE) version 2 to support IP Security (IPSec). Having described the approach, this chapter presents our ongoing efforts to migrate the SSI model to Post-Quantum Cryptography (PQC) as part of the QUBIP project funded by the European Commission. We are at a pivotal moment in cybersecurity. A global community of cryptographers and engineers is working to develop new quantum-secure cryptography and integrate it into existing Internet protocols. The use of PQC will soon become the baseline expectation for security, so it is important to start designing the transition roadmap of the SSI model to contribute to the global effort.

The chapter is structured as follows. First, Section 2 describes the working principles of SSI. This provides the basis for the arguments that follow. Section 3 presents the design of the VC authentication mode in TLS demonstrating the interoperability with existing `X.509` and `RawPublicKey` certificate types. Section 4 presents the design principles for integrating VC into IKEv2 using a seamless approach. Section 5 describes the QUBIP approach to the transition of the SSI model to PQC, which is fully compatible with the adoption of the SSI model in TLS and IKEv2. Finally, Section 6 draws the conclusions and highlights the future research roadmap.

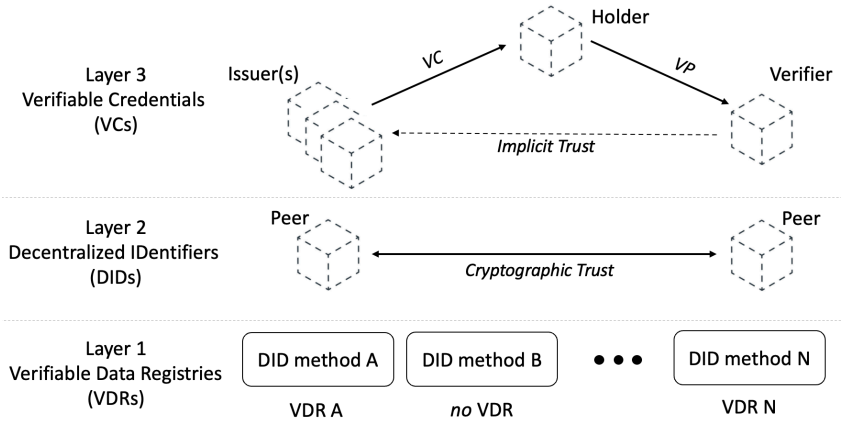

Fig. 1 The self-sovereign identity reference model.

2 Self-Sovereign Identity (SSI)

The SSI reference model [2] consists of three layers, depicted in Figure 1. Each layer contributes to the generation of the identity and defines the basic principles for trustworthy interactions between peers.

Layer 1 is implemented by means of any Verifiable Data Registry (VDR) that is used to store the public identity data. The Decentralized IDentifier (DID) [3] is a new type of globally unique identifier designed to verify a peer. The DID is a Uniform Resource Identifier (URI) in the form

```
did:method_name:method_specific_id
```

where `method-name` is the name of the DID method used to interact with the VDR and `method-specific-id` is the pointer to the DID document stored in the VDR. Thus, a DID associates a peer with a DID document [3] to enable trusted interactions with it. Here is an example of a DID document containing the DID and two verification methods (*i.e.* public keys) for authentication purpose.

```
"id": "did:method_name:method_specific_id",
"authentication": [{
  "id": "did:method_name:method_spec_id#keys-1",
  "type": "JsonWebKey2020",
  "controller": "did:method_name:method_spec_id",
  "publicKeyJwk": { ... encoded public key ... }
  },{
  "id": "did:method_name:method_spec_id#keys-2",
  "type": "JsonWebKey2020",
  "controller": "did:method_name:method_spec_id",
  "publicKeyJwk": { ... encoded public key ... }
  }]
```

The DID method [3] [13] is a software implementation used by a peer to interact with the VDR of choice. Following the W3C recommendation [3], a DID method provides the so-called Create, Resolve, Update, and Deactivate (CRUD) functionalities to

- **Create** a DID: generates an identity key pair (sk_{id}, pk_{id}), the corresponding DID document containing the public key pk_{id} and stores the DID document in the VDR at the `method-specific-id` pointed to by the DID,
- **Resolve** a DID: retrieves the DID document from the `method-specific-id` on the VDR pointed to by the DID,
- **Update** a DID: update the content of the DID document, *e.g.* generates a new key pair (sk'_{id}, pk'_{id}), and stores the modified DID document to the same or a new `method-specific-id` if the node needs to change the DID, and
- **Deactivate** a DID: provides an evidence in the VDR that the DID has been revoked by the owner.

The DID method implementation is VDR-specific and makes the upper layers independent of the VDR of choice. The DID methods [14, 15] using VDRs implemented by a Distributed Ledger Technology (DLT) leverage the immutability features of the DLTs that acts as the Root of Trust (RoT) for DID documents. However, the VDR can also be implemented by a web domain. The `did:web` method [16] leverages a web domain for the RoT of DID documents; with this option, it is up to the implementer to secure the integrity of DID documents according to best practices. Another option exists with `did:key` method [17], in this case the `method-specific-id` components of the DID coincides with the encoded public key, hence the VDR is not required.

Layer 2 uses DIDs and DID documents to establish a cryptographic trust between two peers. In principle, both peers prove the ownership of their private key sk_{id} bound to the public key pk_{id} in their DID document.

While Layer 2 uses DIDs to start authentication, Layer 3 completes it and also deals with authorisation to services and/or resources using Verifiable Credentials (VCs) [4]. A VC is an unforgeable digital credential that contains additional characteristics of the digital identity of a peer than its key pair (sk_{id}, pk_{id}) and DID. A VC contains the metadata to describe properties of the credential (*e.g.* context, id, type, issuer, issuance, and expiration dates), most importantly, the DID and the claim(s) about the identity of the peer in the `credentialSubject` field and the information for any verifier of the VC to check status of revocation of the VC in the `credentialStatus` field. The addition of a proof with the digital signature made by the Issuer of the VC, makes it tamper-evident and trustworthy. Here is an example of a VC, please refer to [4] for detailed explanation of all fields.

```
"@context":
   ["https://www.w3.org/2018/credentials/v1"],
"id": "https://address/credentials/1",
"type": ["VerifiableCredential"],
"issuer": "did:method_name:method_specific_id",
```

```
"issuanceDate": " ... date and time ...  ",
"expirationDate": " ... date and time ...  ",
"credentialSubject": {
    "id": "did:method_name:method_specific_id",
    ... properties to describe the identity ...
},
"credentialStatus": {
    "id": "https://address/credentials/status/#",
    "type": "BitstringStatusListEntry",
    "statusPurpose": "revocation",
    "statusListIndex": " ... index ... ",
}
"proof": {
    "type": " ... type of signature ... ",
    "created": " ... date and time ... ",
    "proofPurpose": "assertionMethod",
    "verificationMethod":
       "did:method_name:method_specific_id#fragment",
    "proofValue": " ... the encoded signature ... "
}
```

The combination of identity key pairs (sk_{id}, pk_{id}), DIDs, and VCs shapes the digital identity of a peer in the SSI ecosystem.

Layer 3 works in accordance with the Triangle-of-Trust depicted in Fig.1. Three different roles coexist:

- **Issuer(s)** asserts claims about a peer, creates a VC from these claims, and issues the VC to the Holder. In addition, they manage VC revocation [4] and provide evidences for the Verifiers to check the revocation status [18] of all issued VCs without compromising the privacy of the holders.
- **Holder** owns one or more VCs and generates a Verifiable Presentation (VP) [4] to authenticate with a Verifier and request a service/resource. A VP is constructed as an envelope of the VC issued by an Issuer, where the proof contains the holder's own signature. Here is an example of VP, please refer to [4] for detailed explanation of all fields.

```
"@context":
   ["https://www.w3.org/2018/credentials/v1"],
"type": "VerifiablePresentation",
"verifiableCredential": [{ ... the VC ... }],
"proof": {
    "type": " ... type of signature ... ",
    "created": " ... date and time ... ",
    "proofPurpose": "authentication",
    "verificationMethod":
       "did:method_name:method_specific_id#fragment",
```

```
        "challenge": " ... challenge from the Verifier ... ",
        "proofValue": " ... the encoded signature ... "
    }
```

- **Verifier** receives a VP from the Holder and verifies the authenticity by checking the signature made by the Issuer on the VC and by the Holder on the VP, checks the revocation status of the VC and the claim(s) and metadata before granting or rejecting access to the Holder. The verifier has implicit trust in the issuer, as represented by the dashed arrow in the Figure 1.

On top of these three layers, it is possible to build any ecosystem of trustworthy interactions among peers, taking advantage of a decentralized architecture.

3 Transport Layer Security (TLS)

TLS [19] is a cryptographic protocol designed to secure communication between a client and a server over the Internet. A secure channel provides (mutual) peer authentication, confidentiality, and integrity of the data exchanged over it. The TLS protocol consists of the TLS Record protocol, which provides connection security, and the TLS Handshake protocol, which allows the client and server to authenticate each other and negotiate security keys before exchanging data.

Following subsections briefly recall the original TLS Handshake protocol (Section 3.1) and, then, present the Handshake protocol with the new `VC` authentication mode (Section 3.2) and its detailed design (Section 3.3).

3.1 Original handshake

The TLS 1.3 handshake protocol establishes the secure channel by exchanging the messages shown in Figure 2. Upon the handshake, client and server negotiate cryptographic parameters through the exchange of `ClientHello` and `ServerHello` messages. Those parameters are the symmetric cipher and hash algorithm used to ensure confidentiality and integrity, respectively. The client and the server generate a secret by using Ephemeral Diffie-Hellman key exchange and use the HMAC-based Extract-and-Expand Key Derivation Function algorithm with the negotiated hash algorithm to derive the session keys from the secret. Then, the server authenticates with the client by sending the `Certificate` message containing the whole certificate chain (Root CA certificate excluded), and the `CertificateVerify` message which is the signature over all the exchanged messages computed with its private key. The client verifies the validity of the certificate chain and the signature in the `CertificateVerify` message to check the identity of the server. The server can request client authentication by sending the `CertificateRequest` message. The client authenticates with the server in the same way.

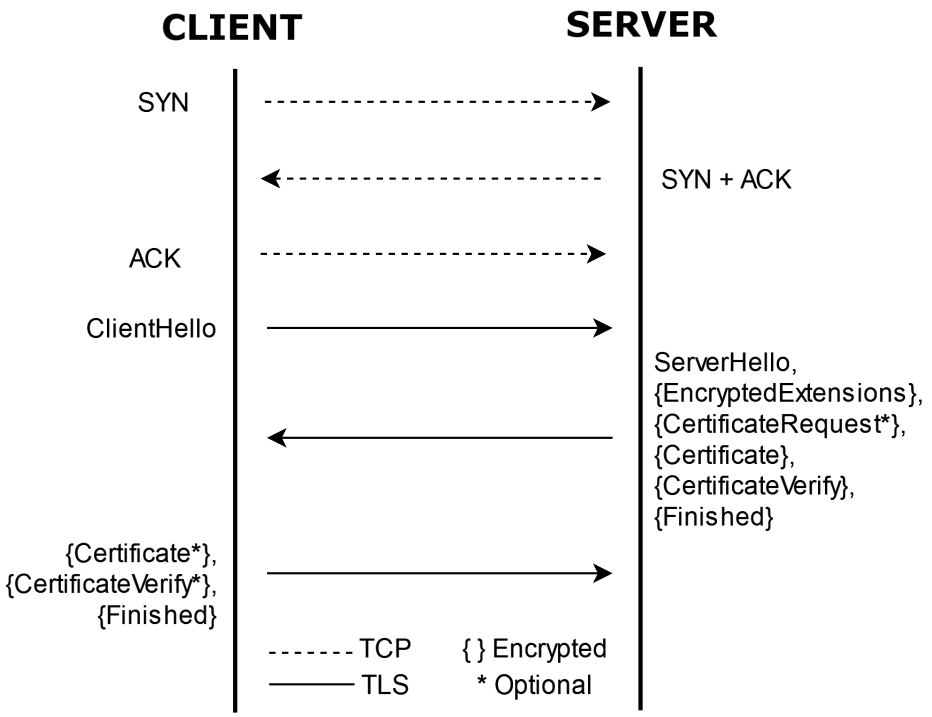

Fig. 2 Original TLS handshake protocol message flow.

3.2 Handshake with VC authentication mode

The main challenges of our new design are to (*i*) introduce a new authentication mode that allows a server (and optionally a client) to authenticate with a peer using VCs, (*ii*) preserve the interoperability with X.509 public key certificates, *i.e.* support hybrid handshake with X.509 certificates and VCs and fallback to the original handshake, and (*iii*) retain all the security features of TLS 1.3.

We met these challenges by leveraging the `client_certificate_type` and `server_certificate_type` extensions introduced by RFC7250 [20] in the original TLS 1.3 handshake. Figure 3 shows the TLS Handshake protocol with `VC` authentication mode.

The client sends these two extensions in the `ClientHello` indicating the certificate types is willing to support for client and server authentication respectively. The server sends the two extensions in the `EncryptedExtensions` message containing the certificate type selected from the list previously sent by the client. This built-in mechanism in the TLS handshake allows the endpoints to exchange different certificate types. We added the `VC` type to the already existing `X.509` and `RawPublicKey` certificate types. A client can authenticate with a VC while the

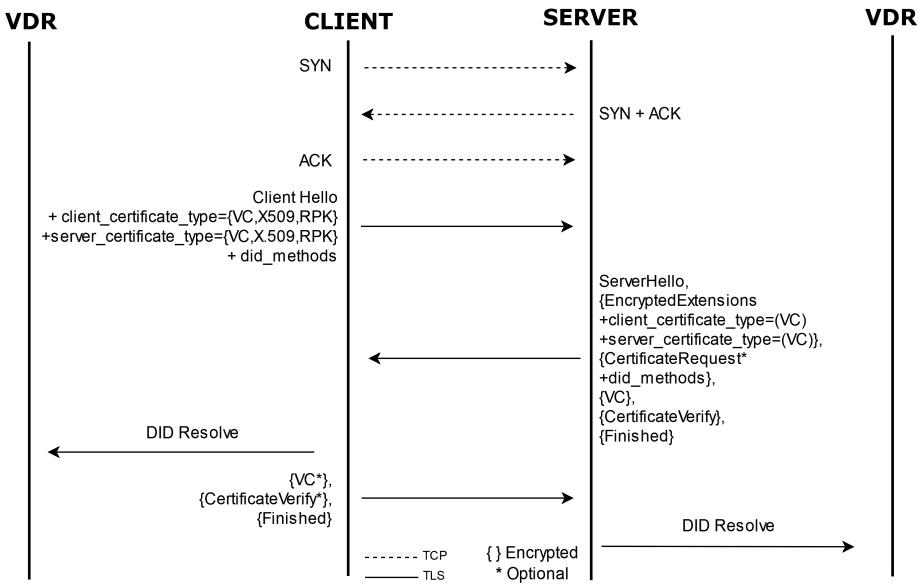

Fig. 3 Message flow of the TLS handshake protocol with `VC` authentication mode.

server using its X.509 certificate. In addition, in the case a server does not support the `client_certificate_type` and the `server_certificate_type` extensions, it can always fall back to the original handshake described in Section 3.1.

We also designed a brand-new extension called `did_methods`. The extension indicates the DID methods an endpoint supports (*i.e.* the ledgers it can interact with) to resolve the peer's DID document. This extension is sent as well in the `ClientHello` and `EncryptedExtensions` messages. A client that sends the `server_certificate_type` extension containing the `VC` certificate type must also send the `did_methods` extension. Similarly, a server that selects `VC` as the certificate type for the `client_certificate_type` extension must also send the `did_methods` extension. A server must send a list of DID methods that both client and server share when it receives the `did_methods` extension from the client.

In Section 2 we stated that a Holder wraps the VC into a VP and signs it before presenting it to a Verifier for authentication. This principle is reproduced correctly in the handshake when TLS endpoints negotiate the `VC` certificate type. The server's `Certificate` message contains the server's VC, so when the server generates the `CertificateVerify` message, it signs all the previous handshake messages, including its VC, with its private key. The randomness in the signature consists of all the handshake messages before the `CertificateVerify` message. This design principle reduces the number of signatures and signature verifications during the handshake while maintaining compliance with the VP concept.

Upon receiving the `CertificateVerify` message, the client checks that the VC follows the scheme specified in the `@context` field, checks the validity of the VC

metadata, resolves the DID of the Issuer and verifies its signature on the VC, and then extracts the server DID from the `credentialSubject` field of the VC and resolves the server DID to retrieve the server public key from the VDR. Finally, the client verifies the signature in the `CertificateVerify` message. Note that in the context of IoT systems, the number of available Issuer(s) is expected to be limited (one very often), therefore both client and server could securely store the public key of Issuer(s) they trust to avoid the DID resolution. The client authenticates with the server in the same way.

3.3 Detailed design

The format of the novel extension and the definition of the new `VC` authentication mode here described retain the same syntax as in RFC8446 [19].

client_certificate_type and server_certificate_type extensions

As defined in RFC7250 [20], the TLS endpoints use the `client_certificate_type` and `server_certificate_type` extensions to negotiate certificate types different from X.509 certificates for authentication. So we have added the `VC` certificate type to the list of types, as shown below:

```
enum {
   X509(0),
   RawPublicKey(2),
   ...
   VC(224),
   (255)
} CertificateType;
```

The list of possible values is maintained by IANA. For initial testing and demonstration purposes, we have assigned the value 224 to `VC`, which is one of the available values reserved for private use.

did_methods extension

This new extension contains a list of DID methods an endpoint supports, *i.e.* a set of VDRs an endpoint can interact with to resolve the peer's DID. A client must send this extension in the extended `ClientHello` message only when it includes the `VC` value in the `server_certificate_type` extension. The server, in the same way, must send this extension in a `CertificateRequest` message only if its `client_certificate_type` extension includes the `VC` value. The

`extension_data` field, is used to carry the `DIDMethodList` structure. The structure of this new extension is shown below:

```
enum {
   name1 (0),
   name2 (1),
   ...
   (65535)
} DIDMethod

struct {
   DIDMethod did_methods<2...2^16-2>
} DIDMethodList
```

The list of existing DID methods is currently maintained by the W3C in [13]. Each DID method is expressed as a string. We propose the DIDMethod enum to map these strings to a two-byte integer value for ease of implementation.

Certificate message

When the selected certificate type is `VC`, the `certificate_list` in the `Certificate` message must contain no more than one `CertificateEntry` with the content of the endpoint's VC as shown below:

```
struct {
   select(certificate_type){
      case VC:
         opaque cert_data<1...2^24-1>;
      ...
   };
   Extension extensions<0...2^16-1>;
} CertificateEntry;

struct {
   opaque certificate_request_context<0...2^8-1>;
   CertificateEntry certificate_list<0...2^24-1>;
} Certificate;
```

4 Internet Protocol Security (IPsec)

After considering the TLS protocol, we now focus our attention on IPsec, in particular aiming to integrate SSI into Internet Key Exchange (IKE). Data exchanged over the Internet is not encrypted by default. IPsec [21] consists of a group of network

protocols responsible for securing communication between two parties at the Internet Protocol (IP) layer of the TCP/IP stack. It provides data origin authentication, data integrity and data confidentiality. IPSec is mainly used to set up end-to-end secure tunnels, and it works by encrypting IP packets, along with authenticating the source where they come from.

The IPSec suite consists of three protocols [22]: Authentication Header (AH), Encapsulating Security Payload (ESP), and IKE. AH and ESP define security features for IP packets, whereas IKE is responsible for establishing a Security Association (SA) between the communicating parties to enable the flow of ESP and AH packets. An SA is a relationship between two parties that describes how they use security services to communicate securely. In IKEv2 [23], all protocol communication consists of pairs of messages in the form of a request (from the Initiator) and a response (from the Responder).

Following subsections briefly recall the original IKEv2 flow of messages (Section 4.1) and, then, present the proposed IKEv2 with VC-based authentication (Section 4.2).

4.1 Original IKEv2 message flow

In IKEv2 [23], the first two pairs of messages `IKE_SA_INIT` and `IKE_AUTH` establish the IKE SA. They authenticate both parties and establish encryption keys, and symmetric and integrity-protection algorithms to secure the data exchange. The authentication can occur through digital signature, Message Code Authentication (MAC) with pre-shared key, or Extensible Authentication Protocol (EAP) [24].

After `IKE_SA_INIT` and `IKE_AUTH`, a third exchange of messages `CREATE_CHILD_SA` can be used for three purposes: (*i*) creating the AH or ESP SAs, called IPSec SAs, (*ii*) re-keying IPSec SAs, and (*iii*) re-keying IKE SAs. The first Child SA will be generated directly through the `IKE_AUTH` exchange without invoking the `CREATE_CHILD_SA`.

Each message is made up of a header (HDR) and a set of payloads. Figure 4 shows the IKEv2 flow of messages, detailing the content of the `IKE_SA_INIT` and `IKE_AUTH` messages. In detail, Figure 4 depicts the case of mutual authentication of parties with certificates.

During the `IKE_SA_INIT` exchange the parties negotiate the cryptographic suite (`SA1`) that includes an encryption algorithm, an integrity-check algorithm and a pseudo-random function (PRF) for key generation, perform the Diffie-Hellman key exchange (`KE`) to generate a shared secret, and exchange a nonce (`N`) for both key generation and protection against the replay attacks. The Responder sends a `CERTREQ` to request a certificate for authentication purposes to the Initiator.

At the end of the first IKEv2 exchange, both parties generate three different symmetric keys: SK_a and SK_e, for integrity protection and encryption of subsequent messages, and SK_d to derive additional key material for Child SAs.

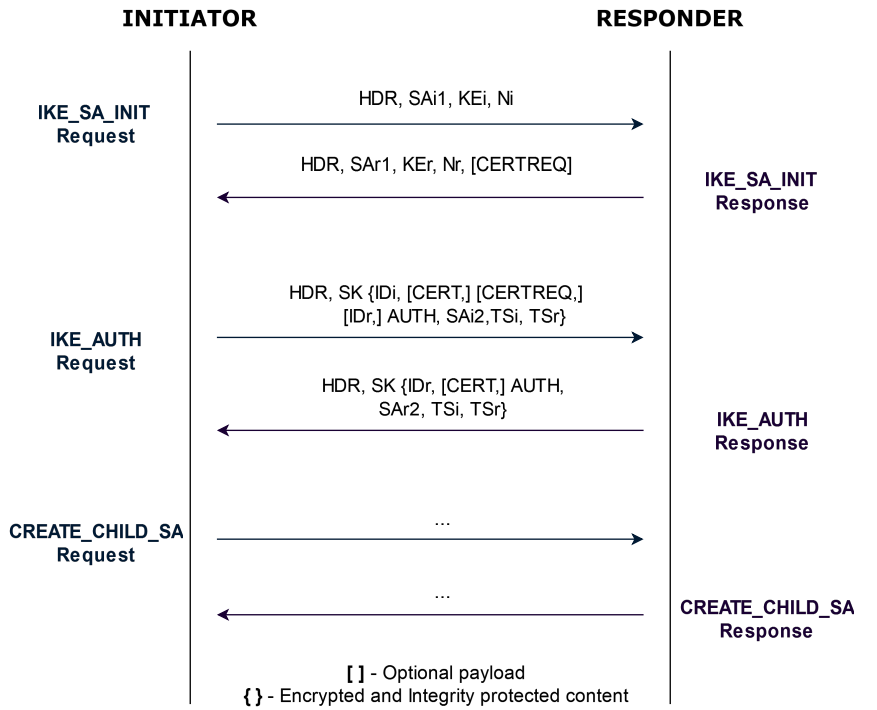

Fig. 4 IKEv2 message flow, where the suffixes `i` and `r` identify the payloads sent by the Initiator and by the Responder, respectively.

During the `IKE_AUTH` exchange the Initiator asserts its identity (`ID`), requests a certificate (`CERTREQ`) from the Responder, sends its certificate(s) (`CERT`), proves its cryptographic identity by signing a block of data with the private key bound to its public key certificate (`AUTH`, see Section 2.15 of [23] for details), and negotiates the first Child SA (`SA2`) for secure traffic exchange. Note that the `ID` is solely used as an internal mechanism to identify SAs and has nothing to do with the one present in the `CERT` payload.

The Responder replies with the same sequence of payloads. It only skips the `CERTREQ` payload that was sent previously. It concludes the negotiation of the Child SA with the `SA2` payload. Traffic selector (`TS`) payload allow both parties to identify packets that should be protected according to the Child SA parameters negotiated.

4.2 IKEv2 with VC authentication

To avoid the management costs of certificates discussed in Section 1, we present here for the first time the modifications to IKEv2 to support authentication with VCs. The new design does not require any new payloads, only a slight modification of two existing payloads, CERTREQ and CERT.

The CERTREQ payload is used to request a certificate to the counterparty. It contains two fields: Cert Encoding to express the certificate type, and Certification Authority to list the trusted anchors for that type of certificate. We add a new VC entry to the list of certificate types for the Cert Encoding field to support Verifiable Credentials. Below is the new list, in which we assign the value 201 to the new type VC, which is the first value available for private use in the IANA list [25]. Of course, the final value will be defined by IANA once the community has reached a consensus.

Certificate Encoding	Value
PKCS #7 wrapped X.509 certificate	1
PGP Certificate	2
DNS Signed Key	3
X.509 Certificate - Signature	4
Kerberos Token	6
Certificate Revocation List (CRL)	7
Authority Revocation List (ARL)	8
SPKI Certificate	9
X.509 Certificate - Attribute	10
Deprecated (was Raw RSA Key)	11
Hash and URL of X.509 certificate	12
Hash and URL of X.509 bundle	13
OCSP Content	14
Raw Public Key	15
VC	201

In our design, when VC is selected as Cert Encoding, the Certification Authority field must contain the list of DID methods that the sender supports to resolve the other party's DID contained in the VC. A two-byte integer can be considered sufficient to represent all the existing DID methods. The list of currently registered DID methods is maintained by the W3C in [13]. Below is the proposed structure for maintaining the mapping.

```
enum {
    name1 (0),
    name2 (1),
    ...
    (65535)
} DIDMethod
```

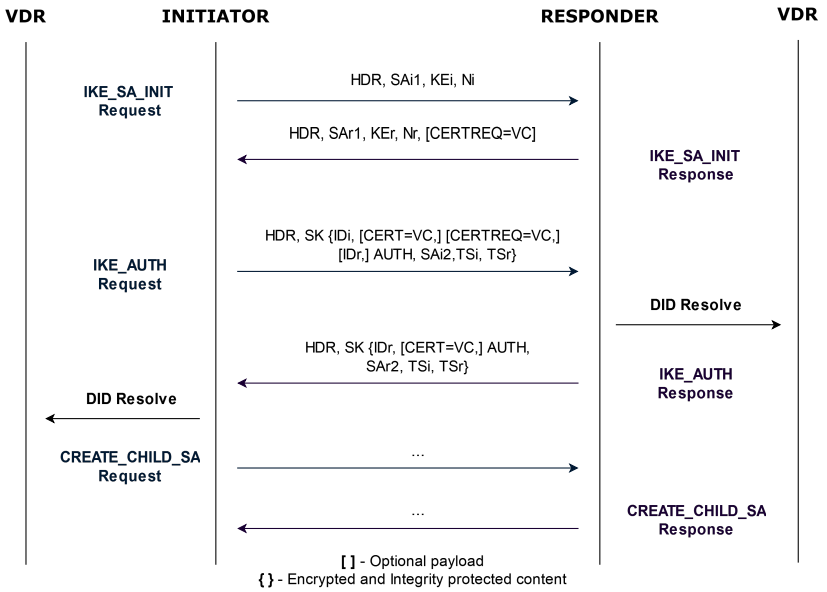

Fig. 5 IKEv2 flow of messages with VC authentication.

In the `Certification Authority` field the first three bytes will indicate the total length of the field, then the next two bytes indicate the length of the list of DID methods and then there is the actual list.

The `CERT` payload carries the party certificate(s) of the type requested in the `CERTREQ` payload. It contains two fields: `Cert Encoding` to express the selected certificate type and the `Certificate Data` that carries the content of the certificate. The list of possible values for the `Cert Encoding` is the same we propose for the `CERTREQ` payload. When `VC` type is selected, the `Certificate Data` field contains the content of the VC. To improve the data transfer efficiency, it is possible to use the Concise Binary Object Representation (CBOR) format [26].

Figure 5 shows the IKEv2 flow of messages in the case of mutual authentication with VCs. The Responder sends the `CERTREQ` payload with the `VC` certificate type and the list of DID methods it supports. During the authentication phase, the initiator checks the availability of the VC for authentication. It then checks that the DID in the VC matches one of the DID methods previously sent by the Responder. If either of these conditions is not met, the Initiator ignores the `CERTREQ` payload and will not send the `CERT` payload without causing an error condition of the IKEv2 protocol, similar to what happens with other certificate types when there is no match. If both of these conditions are met, the Initiator sends the `CERT` payload containing the VC, and the `AUTH` payload with the signature for identity proof over the same content of the original IKEv2 protocol, as discussed in Section 4.1. Note that the Initiator

signs using the private key (sk) associated with the public key (pk) stored in the DID document to which its DID points.

Upon receiving the `IKE_AUTH` Request, the Responder checks (*i*) the VC against schema specified in the `@context` field, (*ii*) the validity of the metadata and the revocation status of the VC, (*iii*) resolves the DID of the Issuer and verifies its signature on the VC, (*iv*) extracts the Initiator's DID from the `credentialSubject` field of the VC and resolves it to the Initiator DID document to retrieve the public key and, finally, (*v*) verifies the signature in the `AUTH` payload. If all checks are successful, the Responder authenticates the Initiator and proceeds with other IKEv2 operations.

The Responder authenticates itself to the Initiator in the same way.

In accordance with the SSI working principles described in Section 2, a Holder wraps the VC in a VP and signs it before presenting it to a Verifier for authentication. In TLS, this behaviour is reproduced by the `CertificateVerify` message, which contains the signature of all previous messages in the handshake, including the VC, see Section 3.2. In IKEv2 this is slightly different. The generation of the signature in the `AUTH` payload does not include the `CERT` payload, which carries the VC. However, the data block signed by both the Responder and the Initiator contains the nonce sent by the other party. This nonce binds the VC to the specific session, replicating the challenge-response mechanism with VPs.

5 Transition to Post-Quantum Cryptography (PQC)

In the light of the new designs presented in Section 3 and Section 4, we have started to look at the transition of the SSI model to post-quantum cryptography (PQC). We are currently working on this topic as part of the QUBIP project funded by the European Commission. QUBIP takes a practical step towards the transition of the SSI ecosystem by designing and implementing the transition to PQC of VC with a pure post-quantum (PQ) and post-quantum/traditional (PQ/T) hybrid approach.

A Cryptographically Relevant Quantum Computer (CRQC) threaten traditional cryptography mainly through the Shor's algorithm [27], for factoring integers and solving discrete logarithms, and Grover's search [28], which can help speed-up unstructured search problems. The main cryptographic technologies in SSI with VC are traditional asymmetric signatures to authenticate identity. As a result, SSI is vulnerable to Shor's algorithm, to the stripping attacks in case of a PQ/T hybrid, but not to the store-now-decrypt-later attacks.

Therefore, the transition to PQC of the SSI ecosystem, with VC, means switching from traditional to PQ or PQ/T hybrid key pair generation, signature generation, and verification. Switching means select and adopt appropriate PQ signature algorithms to be used by the Issuer and Holder to sign the VC and VP respectively, and by the Verifier to verify these signatures for authentication purposes.

The requirements to take into consideration for the selection from the set of PQ signature algorithms under standardisation by National Institute of Standards and Technology (NIST) are different and depends on the application field. In the context

of SSI in IoT systems we defined four main requirements in order of priority: (1) the signature verification time must be as short as possible to increase the number of Holders a Verifier can authenticate, (2) the signature generation time must be as short as possible to increase the number of VCs an Issuer can issue, and to decrease the time a Holder spends preparing the VP, (3) the signature size should be as small as possible to reduce the overall size of the VC and VP, and (4) the public key size should be as small as possible to reduce the size of the DID document.

5.1 Design options

NIST has selected three PQ signature algorithms for standardisation at the end of the third round of the worldwide PQC competition: ML-DSA [29], SLH-DSA [30] and FALCON [31] to be standardised under the name FN-DSA. About 40 new signature algorithms are being evaluated in the current fourth round.

Pure PQ approach

In view of the postponement of FN-DSA standardisation announced by NIST, we have decided to exclude it from the selection at this stage of QUBIP in order to prioritise the PQ signature algorithms, which are closer to final standardisation (expected by the end of 2024 summer). The combined analysis of the requirements described in previous paragraphs, and of the performance evaluation of the PQ signature algorithms selected by NIST, suggests the use of ML-DSA [29], in particular ML-DSA-44 for NIST security level 2, the use of ML-DSA-65 for level 3, and the use of ML-DSA-87 for level 5.

PQ/T hybrid approach

Although the PQ algorithms selected by the NIST have undergone rigorous reviews in recent years, they are not as much mature as the traditional algorithms. This fact has led us to consider hybrid schemes that combine both PQ and traditional signature algorithms in a single cryptographic scheme [32]. The underlying idea is that the hybrid scheme is secure as long as the security of one of the algorithms holds.

In SSI the goal of hybrid signature schemes is hybrid authentication, which is the property that authentication is achieved by the hybrid signature scheme provided that at least one signature algorithm remains secure. In other words, an adversary must violate both schemes to forge a credential and impersonate another user's identity. This way, if the PQ cryptographic assumption is found to be flawed in the future, the composition of the signature algorithms ensures that the scheme is as secure as the traditional signature algorithm.

There are several ways to combine algorithms to build a hybrid scheme [33]. Here, we design a hybrid scheme that combines PQ and traditional signatures using the concatenation combiner and achieving the Weak Non-Separability (WNS) property, customizing the design principles addressed in [34] to the SSI model. The Non-Separability property defined for hybrid signatures in [35] prevents an adversary from removing the signature generated by the secure algorithms, forcing the Verifier to rely only on the signature generated by the broken algorithm. This is commonly referred to as stripping attack. Among the different flavours, the WNS property implies that an adversary cannot remove one of the signatures without the Verifier noticing.

WNS is achieved through the adoption of an artifact. It is the evidence of the will of Issuers and Holders to hybridise their signatures on VC and VP, respectively. Given the SSI working principles introduced in Section 2, our design places the artifact at the protocol level and specifically within the DID document. The artifact coincides with a composite public key object containing the PQ and traditional public keys, and the `AlgID` string that represents the name of the algorithms used to generate the hybrid signature as suggested in [34].

Let us define m the message to be signed as the serialization of the VC or VP, A_{pq} the PQ signature algorithm, A_t the traditional signature algorithm, and (sk_{pq}, pk_{pq}) and (sk_t, pk_t) the PQ and traditional key pairs. The PQ/T hybrid signature $\sigma_h = (\sigma_{pq}, \sigma_t)$ is the concatenation of the PQ signature σ_{pq} and the traditional signature σ_t, calculated as it follows:

$$\begin{aligned} m' &= \text{Hash}(m) \\ \sigma_{pq} &\leftarrow \text{Sign}(sk_{pq}, A_{pq}, \text{DER}(\texttt{OID})||m') \\ \sigma_t &\leftarrow \text{Sign}(sk_t, A_t, \text{DER}(\texttt{OID})||m') \end{aligned}$$

where the Object IDentifier (`OID`) associated to `AlgID` is DER encoded, see Table 1 in [34].

Considering a stripping attack scenario where an adversary has compromised one of the two signature algorithms enough to forge the corresponding signature on the credential, the countermeasure relies on the Verifier only trusting the hybrid public key (pk_{pq}, pk_t) and not the individual components. A Verifier resolving a DID to a DID document with a composite public key must check the whole hybrid signature as it follows:

$$\begin{aligned} m' &= \text{Hash}(m) \\ \text{Verify}(pk_{pq}, \text{DER}(\texttt{OID})||m', \sigma_{pq}, A_{pq}) \\ \text{Verify}(pk_t, \text{DER}(\texttt{OID})||m', \sigma_t, A_t) \end{aligned}$$

where the `OID` is selected based on the `algID` retrieved from the DID document.

The requirements and principles for the selection of the PQ signature algorithms previously discussed also apply in the case of a PQ/T hybrid approach to the transition. In addition, the authors of [34] provide guidelines to compose signature

algorithms based on their bit-level security and to select the appropriate pre-hash to be applied to the message m.

All this information together suggests the adoption of two hybrid schemes using of ML-DSA and Ed25519. In particular, the schemes with `algID` equal to id-MLDSA44-Ed25519-SHA512 and id-MLDSA65-Ed25519-SHA512. The former composes ML-DSA-44 with Ed25519, while the latter composes ML-DSA-65 again with Ed25519. In both cases, the authors of [34] suggests using SHA512 as the pre-hash for consistency with Ed25519's internal use of this digest algorithm.

6 Conclusions

IoT is the second trend driving the deployment of applications using PKIs, meaning that the digital identity is increasingly fundamental in these connected systems. However, it is widely believed that the use of PKI with certificates is not the most appropriate solution for managing identities in large-scale deployments. The main problem is the high cost of managing certificates throughout their lifecycle, from installation to regular updates and revocation. SSI provides an alternative solution to PKI to reduce the complexity and the associated costs of certificate management. An SSI-native IoT reduces the need for human intervention in identity management, significantly reducing the complexity and cost of identity management in large-scale deployments. The chapter has presented an alternative approach that integrates SSI below the application layer of the TCP/IP. The SSI has been integrated into the TLS handshake protocol and into IKEv2 to provide an efficient alternative to certificates while maintaining interoperability with existing identity models (*e.g.* X.509 certificates and raw public keys). We believe in this approach, which is why we have also started the transition of SSI to PQC. The next step is to discuss our design choices in the IETF to open the discussion with the industry community.

Acknowledgements This work has been partially developed within the QUBIP project, funded by the European Union under the Horizon Europe framework programme [grant agreement no. 101119746] (https://www.qubip.eu), and the SEDIMARK project, funded by the European Union under the Horizon Europe framework programme [grant agreement no. 101070074] (https://sedimark.eu).

References

1. K. R. Falls, W. R. Stevens, TCP/IP Illustrated, Volume 1: The Protocols, Second Edition. Boston, USA: Addison-Wesley, 2011.
2. A. Preukschat, D. Reed, Self-Sovereign Identity – Decentralized digital identity and verifiable credentials. Shelter Island, NY: Manning, 2021. Available: https://www.manning.com/books/self-sovereign-identity
3. W3C, Decentralized Identifiers (DIDs) v1.0. Core architecture, data model, and representations. W3C Recommendation, 2022. https://www.w3.org/TR/did-core/

4. W3C, Verifiable Credentials Data Model v2.0. W3C Candidate Recommendation Draft, 2024. https://www.w3.org/TR/vc-data-model-2.0/
5. S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, D. Cooper, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 5280, 2008. https://www.rfc-editor.org/info/rfc5280
6. J. Won, A. Singla, E. Bertino, G. Bollella, Decentralized Public Key Infrastructure for Internet-of-Things, Proceedings of IEEE Military Communications Conference, 2018, pp. 907–913.
7. M. Pahl, L. Donini, Giving IoT Services an Identity and Changeable Attributes, Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management, 2019, pp. 455–461.
8. H. Wan, Q. Wang, Y. Teng, C. Ma, J. Lin, M. Wang, ImCT: A Feasible Scheme for Deploying Implicit Certificates with Certificate Transparency in IoT, Proceedings of the IEEE International Conference on Computer Communications and Networks, 2023, pp. 1–10.
9. J. Höglund, S. Raza, LICE: Lightweight Certificate Enrollment for IoT using Application Layer Security, Proceedings of the IEEE Conference on Communications and Network Security, 2021, pp. 19–28.
10. C. Boudagdigue, A. Benslimane, A. Kobbane, J. Liu, Trust-Based Certificate Management for Industrial IoT Networks, IEEE Internet of Things Journal, 10 (14), 2023, pp. 12867–12885.
11. J. Höglund, S. Lindemer, M. Furuhed, S. Raza, PKI4IoT: Towards Public Key Infrastructure for the Internet of Things, Computers & Security, 89, 2020.
12. D. E. Majdoubi, H. E. Bakkali, M. Bensaih, S. Sadki, A Decentralized Trust Establishment Protocol for Smart IoT Systems, Internet of Things, 20, 2022.
13. W3C, DID Specification Registries. The interoperability registry for Decentralized Identifiers. W3C Group Note, 2023. https://www.w3.org/TR/did-spec-registries/
14. IOTA Foundation, Digital Identity, 2024. https://www.iota.org/solutions/digital-identity
15. ZKID labs AG, Digital Identity, 2024. https://www.polygonid.com
16. W3C, The did:web Method Specification. 2023. https://w3c-ccg.github.io/did-method-web/
17. W3C, The did:key Method Specification. A DID Method for Static Cryptographic Keys, 2022. https://w3c-ccg.github.io/did-method-key/
18. W3C, Bitstring Status List v1.0: Privacy-preserving status information for Verifiable Credentials. W3C Working Draft, 2024. https://www.w3.org/TR/vc-bitstring-status-list/
19. E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, RFC 8446, 2018. https://www.rfc-editor.org/info/rfc8446
20. P. Wouters, H. Tschofenig and J. Gilmore, S. Weiler, T. Kivinen, Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS), RFC 7250, 2014. https://www.rfc-editor.org/info/rfc7250
21. S. Frankel, S. Krishnan, IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap, RFC 6071, 2011. https://www.rfc-editor.org/info/rfc6071
22. K. Seo, S. Kent, Security Architecture for the Internet Protocol, RFC 4301, 2005. https://www.rfc-editor.org/info/rfc4301
23. C. Kaufman, P. E. Hoffman, Y. Nir, P. Eronen, T. Kivinen, Internet Key Exchange Protocol Version 2 (IKEv2), RFC 7296, 2014. https://www.rfc-editor.org/info/rfc7296
24. J. Vollbrecht, J. Carlson, L. Blunk, B. Aboba, H. Levkowetz, Extensible Authentication Protocol (EAP), RFC 3748, 2004. https://www.rfc-editor.org/info/rfc3748
25. IANA, Internet Key Exchange Version 2 (IKEv2) Parameters. https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml
26. C. Bormann, P. E. Hoffman, Concise Binary Object Representation (CBOR), STD 94, RFC 8949, 2020. https://www.rfc-editor.org/info/rfc8949
27. P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM Journal on Computing, 26(5), 1997, pp. 1484–1509. [Online]. Available: https://doi.org/10.1137/S0097539795293172
28. L. K. Grover, A fast quantum mechanical algorithm for database search, Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, New York, NY, USA: ACM 1996, p. 212–219.

29. NIST, Module-Lattice-Based Digital Signature Standard, FIPS 204, August 2023. https://doi.org/10.6028/NIST.FIPS.204.ipd
30. NIST, Stateless Hash-Based Digital Signature Standard, FIPS 205, August 2023. https://doi.org/10.6028/NIST.FIPS.205.ipd
31. P. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang, Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU v1.2, October 2020. https://falcon-sign.info
32. N. Bindel, B. Hale, D. Connolly, and F. Driscoll, Hybrid signature spectrums, Internet-Draft, draft-hale-pquip-hybrid-signature-spectrums-04, 2024, Work in Progress. https://datatracker.ietf.org/doc/draft-hale-pquip-hybrid-signature-spectrums/04/
33. A. Pino, D. Margaria, A. Vesco, On PQ/T Hybrid Verifiable Credentials and Presentations to Build Trust in IoT Systems, Proceedings of IEEE International Conference on Smart and Sustainable Technologies. Bol and Split, Croatia: IEEE, 2024, pp. 1–6.
34. M. Ounsworth, J. Gray, M. Pala, and J. Klaußner, Composite ML-DSA for use in Internet PKI, Internet-Draft, draft-ounsworth-pq-composite-sigs-13, 2024, Work in Progress. https://datatracker.ietf.org/doc/draft-ounsworth-pq-composite-sigs/13/
35. N. Bindel, U. Herath, M. McKague, D. Stebila, Transitioning to a quantum-resistant public key infrastructure, Cryptology ePrint Archive, Paper 2017/460, 2017. https://eprint.iacr.org/2017/460