FYI - the first round of comment review on this working draft was completed on Feb 7, 2024. My plan is to then work on a revision and at the same time migrate to GitHub to facilitate better discussions. Meanwhile, while you are welcome to continue providing comments here, I will not respond to new comments arriving after Feb 7 on this draft, but postpone them to the new revision when it's ready in GitHub. Thanks for all of your support. I hope this transition is relatively short in duration.

Trust Spanning Protocol

1. Overview

The Trust Spanning Protocol (TSP) facilitates secure communication between endpoints with potentially different identifier types, using message-based exchanges. As long as these endpoints use identifiers based on public key cryptography (PKC) with a verifiable trust root, TSP ensures their messages are authentic and, if optionally chosen, confidential. Moreover, it presents various privacy protection measures against metadata-based correlation exploitations. These attributes allow endpoints to form authentic relationships rooted in their respective verifiable identifiers (VIDs), viewing TSP messages as virtual channels for trustworthy communication.

Beyond offering enhanced trust properties compared to previous solutions, the TSP is conceived as a universal protocol, serving as a foundation for various higher-layer protocols. This design approach draws inspiration from the success of the TCP/IP protocol suite. In the TSP context, directional TSP messages function as a unified primitive to bridge diverse endpoint types, similar to how IP packets enable inter-networking between distinct networks. Trust task protocols, intended to operate atop of the TSP, mirror the roles of TCP or UDP, providing trust task-specific solutions while harnessing the core properties of the TSP.

TSP messages, serving as general-purpose primitives, can be transported directly between endpoints or routed via intermediaries. This specification first focuses on the direct mode, then describes the routing mechanism in Section 6.

Furthermore, TSP messages can be either between two endpoints (unicast) or among multiple endpoints (multicast or broadcast). Two methods for relatively simple but common multicast and broadcast scenarios are detailed in Section 7. Additional methods may be defined separately in the future.

TSP messages can traverse various transport mechanisms without prior assumptions as to their trustworthiness. While we may select specific underlying transport protocols for TSP based on various factors, including additional security considerations, it's crucial to note that endpoint identifiers function as conceptual addresses for TSP messages. These identifiers must facilitate address resolution procedures, ensuring TSP's capability to transport and deliver messages effectively.

TSP stands as the foundational spanning layer protocol within the Trust over IP technology architecture [X add reference]. It occupies a pivotal role, facilitating the concurrent achievement of robust trust properties and universal interoperability across the Trust over IP stack. For additional details on the architecture, please see [X add reference] and Section 1.2 below.

1.1 Requirement Notations

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2 Reference Architecture

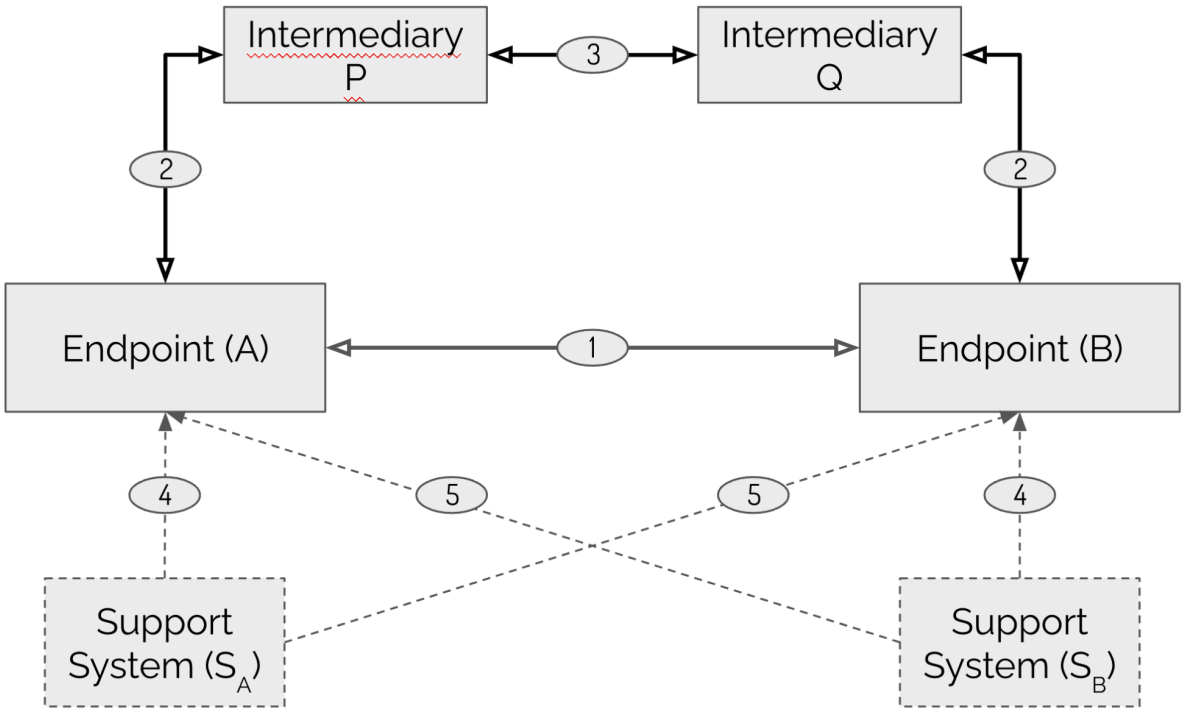

Figure 1: TSP Reference Architecture

The Trust Spanning Protocol is defined within the Reference Architecture (RA) illustrated in Figure 1. The principal components of this reference architecture are:

- Direct Communication: Endpoints communicate with each other using the TSP in direct mode, depicted by an arrowed line labeled number 1. This communication pattern encompasses two directional relationships, with each endpoint evaluating the trustworthiness of the other independently.

- Routed Communication: Endpoints communicate using the TSP in routed mode through intermediaries, represented by arrowed lines labeled numbers 2 and 3. It's important to note that intermediaries are not necessarily trustworthy.

- Identifier Management: Endpoints manage their verifiable identifiers (VIDs) and associated roots of trust information via an abstract interface with their support systems, shown by dotted lines labeled number 4. Additionally, endpoints verify and assess the counterpart in a TSP relationship through another abstract interface with their respective support systems, denoted by dotted lines labeled number 5.

1.3 Authenticity, Confidentiality, and Metadata Privacy

In TSP, these properties are defined within the context of a directional relationship formed by a pair of verifiable identifiers between a source and a destination endpoint. In this context, the source is also referred to as the sender and the destination as the receiver of a message. Authenticity is ascertained by the receiver, providing confidence that the received message remains unaltered and that the message genuinely originates from the sender. Confidentiality ensures that only the sender and receiver have access to the protected confidential payload data content. However, some parts of the message's envelope, not shielded by confidentiality protection, can be observed and used to infringe upon privacy through traffic analysis, correlation or other exploitative means. TSP provides optional mechanisms to safeguard against these vulnerabilities. This specific type of protection is termed "metadata privacy," differentiating it from the narrower understanding of privacy, which concerns the prevention of content exposure to unauthorized parties, synonymous with confidentiality.

TSP messages always assure authenticity, optionally confidentiality, and if utilized, metadata privacy.

1.4 Use of Formats

TSP specifies message types that will have varying formats or representations during their lifecycle, both within systems that process or store them and networks that transport them. Additionally, for purposes such as debugging, documentation, or logging, these messages might need to be represented in a text format that is more accessible for human interpretation. In this specification, we utilize text formats for clarity and illustrative purposes. However, it's essential to understand that such text-based descriptions are solely to illustrate how the messages are structured in binary format and encoded for transport. For complete details on serialization and encoding, please refer to Section 7.

2. Verifiable Identifiers

The Trust Spanning Protocol does not mandate that endpoints utilize only a single type of identifier and this specification does not define one. However, the efficacy of TSP and the trust assurances in authenticity,

confidentiality, and contextual privacy it provides hinge on the methodologies of VIDs. Factors such as the construction and resolution of these identifiers, coupled with the verification of trust information from their support systems, directly influence the degree of trust endpoints can derive from using TSP. In this section, we outline high-level requirements without prescribing how various VID types should fulfill them. All identifiers that meet these standards are termed Verifiable Identifiers (VID). The aim is to enable endpoints, equipped with their chosen VID type, to communicate over TSP with the respective confidence and trust level that the VID inherently supports.

A foundational prerequisite for TSP is that endpoints operate within a secure and trustworthy computing environment, possibly facilitated by tools such as Trusted Execution Environments (TEEs), digital wallets, or digital vaults. While TSP aids in transmitting trust signals between endpoints, it cannot instantiate trust where none exists.

In TSP, pairs of TSP endpoints establish directional relationships. In these relationships, endpoints assess each other's identifiers independently. The verification and evaluation of VIDs remain inherently directional.

2.1 VID Use Scenarios

In the Trust Spanning Protocol, VIDs function as identifiers within protocol envelopes and other control fields (see Section 3). As identifiers in exposed envelopes, VIDs may be visible to third parties with access to the network transports, allowing for potential correlation with other identifying transport mechanism information, such as IP addresses, transport protocol header information, and other metadata like packet size, timing, and approximate location. To mitigate the risk of metadata exploitation, TSP provides Nested Messages (Section 4) and Routed Messages (Section 5) for certain metadata privacy protections. Given the varied roles VIDs play in different scenarios, their management requires distinct considerations. To clarify and simplify the discussion of these scenarios, we categorize VID uses into three types: public, well-known, and private.

We refer to the scenarios where VIDs are exposed to external entities as their 'public use'. The address resolution operations of public use VIDs may provide visible information to an adversary.

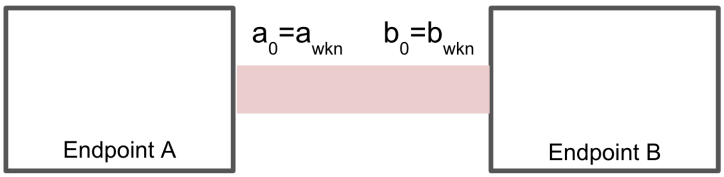

Figure 2: An example of public use VIDs where a_0 and b_0 are used in the first layer above a public transport mechanism. Additionally, a_0 and b_0 may also be made 'Well-Known', labeled as a_{wkn} and b_{wkn}.

It's important to note that while additional security measures like TLS or HTTPS can be employed at the transport layer to safeguard VIDs, TSP does not inherently depend on these mechanisms for protection. Consequently,

within the context of TSP, even VIDs protected by such transport layer security are treated as if they are 'public,' assuming they could potentially be accessed or observed by external parties.

Within the category of public use VIDs, there is a subclass known as 'Well-Known VIDs'. These are VIDs whose controllers deliberately intend for them to be broadly recognized. The rationale behind making a VID well known often revolves around streamlining or simplifying the processes of VID discovery, resolution, and verification. However, it's important to recognize that such actions inherently expose additional information to potential adversaries. Figure 2 above also shows an example of Well-Known VIDs. As a subclass of public VIDs, well-known VIDs must also meet all public VID requirements.

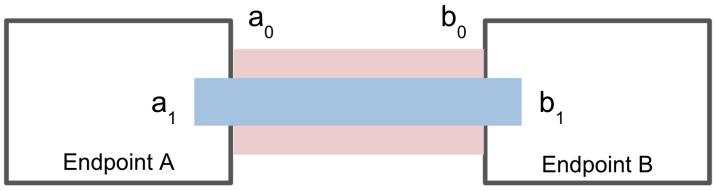

Figure 3: VIDs a_0 and b_0 are in public use; VIDs a_1 and b_1 are in private use.

VIDs are considered to be in 'private use' when their usage is safeguarded within another instance of TSP relationship through a nested approach (See Section 4). This private utilization is facilitated by nesting, where the inner VIDs bypass the need for address resolution. Their establishment operations are managed by TSP control messages, and all relevant operations are secured by the encryption provided by the outer layer of TSP. For an in-depth explanation of these nested modes of TSP, please refer to Section 4. The specifics regarding control messages are detailed in Section 7.

2.2 VID General Requirements

2.2.1 Cryptographic Non-Correlation

An endpoint can control multiple VIDs simultaneously and over extended periods. It is imperative that these VIDs are cryptographically non-correlatable in an information-theoretic security context, meaning the knowledge of one VID does not reveal any information about another.

For example, in Figure 3, an adversary might observe VIDs a_0 and b_0, which are categorized as public and could be linked to a specific endpoint using additional metadata. However, if the same adversary also happens to observe VID a_1, it should be impossible, based on the design of the VIDs, for the adversary to establish a correlation between a_1 and a_0, and consequently, to associate a_1 with endpoint A.

2.2.2 Verification

VIDs for public use MUST be based on and be verifiable through Public Key Cryptography (PKC). If endpoint '**a**' uses `VID_a` as one of its identifiers, any party evaluating `VID_a` should be able to resolve `VID_a` to its associated public key for the purpose of verification, `PK_a`. Furthermore, they should be able to verify that endpoint '**a**' has access to the corresponding secret key, `SK_a`, using a PKC algorithm. These verification processes inevitably depend on the specific design and implementation of each identifier, as well as the trust information source they utilize. In many instances, this trust source can't be independently verified by an endpoint. In such cases, it MAY be appraised through other methods that satisfy the endpoint within its application context.

Endpoints MUST execute verification procedures as required corresponding to each type of public VID upon initial use and whenever any changes that necessitate re-verification.

For any VID that is for private use, while the same verification procedure requirements above still apply, they MAY use much simpler VID types because its verification is between two endpoints who already have a verified TSP relationship between them and the verification is conducted through trusted TSP messages of that verified relationship. TSP defines message types for such cases of private VID verification.

Ultimately, whether an endpoint accepts the verification result as part of its broader acceptance criteria depends on the endpoint and the application context. Endpoints must always take into account the governance frameworks associated with the specific VID type, which may not be fully assessable through purely cryptographic means or in real-time. The verification requirements specified above should not be seen as exhaustive.

2.2.3 Resolution to Transport Address

For every VID to be in public use, an address resolution mechanism is necessary. This mechanism MUST be capable of taking a VID as input and producing a transport or other communication layer access point, ensuring the successful delivery of messages. The specific details of this procedure will, of course, vary based on the transport mechanism in use.

For any VID that is used in private only, an address resolution mechanism is unnecessary.

2.2.4 Handling Changes

VIDs may require handling of key rotations, replacement of intermediaries, and other types of changes over the lifetime of a VID. If such changes occur, then the endpoints MAY be required to re-verify or refresh the dependent information before they use them again.

2.2.5 W3C DIDs

The TSP reference architecture sets out an endpoint-to-endpoint communication pattern, aligning directly with the model defined in W3C DID Core [x]. VIDs and DIDs have overlapping characteristics, including but not limited to namespace requirements, URI compliance, and the resolution method to obtain definitive information.

If a VID type is a DID, it SHOULD comply with W3C DID Core [x].

VIDs are not limited to being DIDs. Additionally, not all DIDs qualify as VIDs. For a DID to be recognized as a VID, it must meet specific additional requirements listed in the following sections.

2.3 VID Appraisal Framework

Before an endpoint initiates a send or receive action involving a VID, it must evaluate the trust foundation of that VID. It's crucial to understand that this assessment pertains to the evaluation of the identifier itself and not the controller or the subject of the identifier. While the verification procedure (as detailed in Section 2.1.2) forms a portion of this evaluation, it might not provide comprehensive assurance for certain applications.

TODO

2.4 VID Implementations and Considerations

TODO: choose which examples to include. The purpose of this section is for informational illustration only. The examples are by no means exclusive.

2.4.1 did:keri

TODO

2.4.2 did:webs

TODO

2.4.3 Autonomic ID

TODO

2.4.4 X.509 based VID

TODO

2.4.5 OIDC

TODO

3. Messages

TSP operates as a message-based communication protocol. The messages in TSP are asynchronous, can vary in length, and are inherently directional. Each message has a designated sender (alternatively termed "source") and a receiver (or "destination"). Throughout this specification, in particular when we describe the routed mode in Section 6, the terms "sender" and "receiver" will be used to refer to direct neighbors, while the terms "source" and "destination" will be used for the originating and ending endpoints of the carried message. Within the context of TSP, both the sender and the receiver of a message qualify as "endpoints." Entities such as Intermediaries or Support Systems can also function as endpoints when they are participating in TSP communications themselves. For the sake of simplicity, we will uniformly refer to all these entities as "endpoints," unless a distinction is necessary for clarity.

As outlined in Section 2, VIDs serve as identifiers for any entities involved in TSP. Both the sender's VID and the receiver's VID fulfill the dual roles of identifier root of trust verification and resolution to a transport address for delivering the TSP message. The sender and receiver VIDs can be of different VID types.

TSP messages are made of three parts: envelope, payload and signature, as illustrated in the pseudo-formula and the accompanying Figure 4 below.

```
TSP_Message = {Envelope, Payload, Signature}
```

The notation "`{a, b, c}`" defines a string level concatenation. For more information on `Concat`, please refer to Section 8.

Figure 4: TSP Message

We now define these parts in the following sections.

3.1 TSP Envelope

A TSP message is enclosed within a standardized envelope structure, ensuring consistent formatting and interpretation. The layout of this envelope is graphically represented in the subsequent diagram and is explained further in the subsequent text.

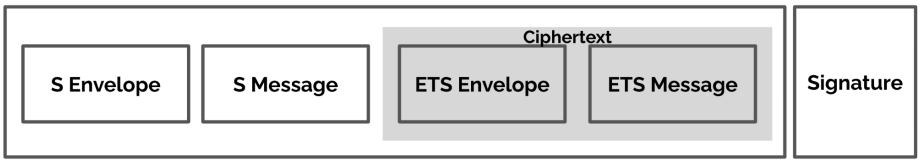

Figure 2: TSP Message Envelope

The TSP message structure is built upon the ESSR (Encrypt Sender Sign Receiver) authenticated encryption approach, using public-key cryptography, as first introduced in [2]. However, there are several modifications and extensions to better suit the TSP's needs.

Referencing Figure 2, the TSP message is composed of three sections:

1) Signed Section: This section is in plaintext.
    a) Signed Envelope (S Envelope)
    b) Signed Message (S Message): Optional.
2) Encrypted then Signed (ETS) Section: This section is in ciphertext.
    a) ETS Envelope
    b) ETS Message: Optional.
3) The concatenated two sections are then signed to generate the Signature.

In TSP, rather than directly using the public keys as seen in ESSR, VIDs are utilized. In adherence to the ESSR scheme, the S Envelope MUST contain the receiver's VID, and the ETS Envelope MUST include the sender's VID.

Additionally, we require that the S Envelope MUST also contain the sender VID, and MAY incorporate an optional message body (S Message) that is signed but not encrypted.

```
TSP_Message = {Cleartext, Ciphertext, Signature}
            = {S_Envelope, S_Message, Ciphertext, Signature}
```

The notation "`{a, b, c}`" defines a string level concatenation. For more information on `Concat`, please refer to Section 8.

```
{a, b, c} = Concat(a, b, c).
```

The Ciphertext is encrypted using the receiver's public-key over the ETS Envelope and ETS Message.

$$\texttt{Ciphertext} = \texttt{Encrypt}_{\texttt{VID\_rcvr.PK}}(\{\texttt{ETS\_Envelope, ETS\_Message}\}).$$

The Signature is generated using the sender's secret-key over both the S part of the message and the Ciphertext of the ETS part of the message.

$$\texttt{Signature} = \texttt{Sign}_{\texttt{VID\_sndr.SK}}(\{\texttt{S\_Envelope, S\_Message, Ciphertext}\}).$$

The general format outlined here can be termed as an "envelope". Within this envelope, there are two designated locations for placing message content: the S and ETS Message bodies.

The algorithms used for both `Encrypt` and `Sign` are detailed in Section 8.

Note that the encryption key VID_rcvr.PK (public key) and signing key VID_sndr.SK (secret key) are defined in the following Section 3.2.

In some legal settings, a cryptographic signature applied to a ciphertext may lack the same contractual force as a signature applied directly to the cleartext. In such cases, a signature may be added to the message itself. This field is a part of the Message, not the Envelope, and is described in Section 3.6.

3.2 S Envelope

The S Envelope is signed but not encrypted and is visible to non-participating third-party observers.

```
S_Envelope = {
     VID_sndr,        //Required
     VID_rcvr,        //Required
     …
}
```

The observers may verify the authenticity of the sender by using the cleartext VID_sndr and the Signature. Such observers may use this ability for legitimate purposes, e.g., to thwart DoS attacks by infrastructure equipment, but also for undesirable exploitations from the endpoints' perspective.

The S Envelope may be extended to include additional information as shown above and defined later in this specification. The first two fields however will always be VID_sndr and VID_rcvr.

3.3 ETS Envelope

The ETS Envelope (together with ETS Message) is encrypted into Ciphertext and then signed. It can only be read by the receiver after decryption with its secret-key.

```
ETS_Envelope = {
     VID_sndr,        //Required
     …
}
```

The ETS Envelope may be extended to include additional information in the future. The first field however will always be VID_sndr.

3.4 A Shorthand Notation

A simple shorthand notation can be used without ambiguity for the general TSP message envelope. This notation will be detailed and expanded as we address additional aspects of TSP.

[VID_sndr, VID_rcvr, ETS_Message] = { S_Envelope, S_Message, Ciphertext, Signature },

Where,

S_Envelope = {VID_sndr, VID_rcvr, … },
ETS_Envelope = {VID_sndr, …},
Ciphertext = $\text{Encrypt}_{\text{VID\_rcvr.PK}}$ (ETS_Envelope, ETS_Message),
Signature = $\text{Sign}_{\text{VID\_sndr.SK}}$ (S_Envelope, S_Message, Ciphertext).

3.5 Envelope Examples

In the following subsections, we briefly overview a few common usages of the TSP message envelope structure and discuss their trust properties. Naturally, TSP message uses are not limited by these examples.

3.5.1 Authentic Confidential Message

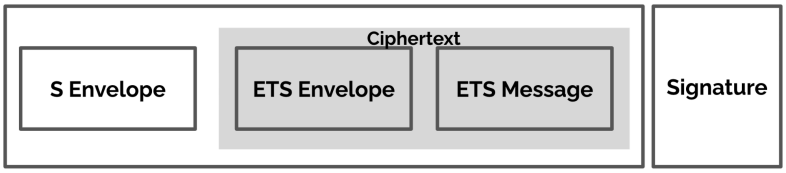

Figure 3: Authentic Confidential Message Envelope

Authentic-Confidential-Message =
{VID_sndr,
VID_rcvr,
Ciphertext,
Signature}

Ciphertext = $\text{Encrypt}_{\text{VID\_rcvr.PK}}$({VID_sndr, ETS_Message})

Signature = $\text{Sign}_{\text{VID\_sndr.SK}}$({VID_sndr, VID_rcvr, Ciphertext})

This envelope structure uses ETS Message only.

The sender is assured that

- Only the receiver that controls VID_rcvr can decrypt and read the ETS Message. (Confidentiality)
- The receiver knows that the message is from the sender that controls VID_sndr AND the message has not been tampered. (Authenticity, or Third-party Unforgeability))
- The receiver cannot convince a third-party with forged sender, receiver or message. (Receiver Unforgeability)

The receiver is assured that

- The message is from the sender that controls VID_sndr AND the message has not been tampered. (Authenticity)
- Only it can decrypt and read the ETS Message. (Confidentiality)

Infrastructure nodes of the sender

-

Infrastructure nodes of the receiver

-

The Authentic Confidential Message (ACM) envelope is a common envelope used throughout this specification. A shorthand notation is useful for brevity and clarity and is defined below:

[VID_a, VID_b, Msg] = { VID_a, VID_b, Ciphertext, Signature },

Where,

Ciphertext = $\text{Encrypt}_{\text{VID\_b.PK}}$ (VID_a, Msg),

Signature = $\text{Sign}_{\text{VID\_a.SK}}$ (VID_a, VID_b, Ciphertext).

The above formula is definitive without ambiguity for ACM envelopes. It can also be extended for other envelope types, for example, for nesting (Section 4), for routing (Section 6) or for multicast (Section 7). Please refer to those sections for further details.

3.5.2 Authentic Non-Confidential Message

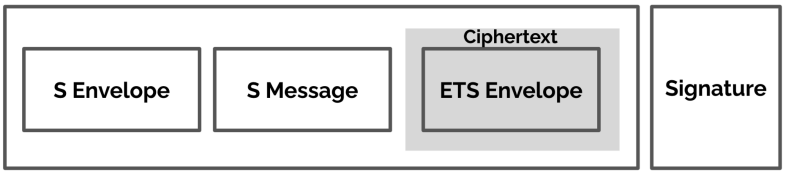

Figure 4: Authentic but Non-Confidential Message Envelope

```
Authentic-Non-Confidential-Message =
      {VID_sndr,
       VID_rcvr,
       S-Message,
       Ciphertext,
```

Signature}

$Ciphertext = Encrypt_{VID\_rcvr.PK}(\{VID\_sndr\})$

$Signature = Sign_{VID\_sndr.SK}(\{VID\_sndr, VID\_rcvr, S\text{-}Message, Ciphertext\})$

This envelope structure uses S Message only. The sender intends the message to be non-confidential. The ciphertext contains only the encrypted `VID_sndr` but it is necessary to defend against a third-party who may otherwise forge the sender VID.

The receiver of such messages is assured that

- The receiver knows the message is from the sender that controls VID_sndr AND the message has not been tampered. (Authenticity, or Third-party Unforgeability))
-

3.6 Relationships

When endpoint **A**, identified by `VID_a` (sender), sends a TSP message to endpoint **B**, identified by `VID_b` (receiver), a TSP relationship is established: `<VID_a, VID_b>`. For endpoint **A**, this relationship is represented as a directional mapping from its local VID (`VID_a`) to the remote VID (`VID_b`). Conversely, when endpoint **B** receives and verifies the message envelope, it creates a relationship in the opposite direction: `<VID_b, VID_a>`. Endpoint **B** establishes this relationship provided it successfully verifies and appraises `VID_a`.

Endpoint **B** has the option to send a message back to **A** using the same pair of VIDs. This message could either contain user data or simply act as a control message to notify endpoint **A**. By sending this return message from endpoint **B** to **A**, the previously directional relationships become bidirectional pairs: `(VID_a, VID_b)` in endpoint **A** and `(VID_b, VID_a)` in endpoint **B**.

Notationally, we use `<a, b>` to indicate a directional relationship with `'a'` as the local endpoint and `'b'` as the remote, and `(a, b)` to indicate bidirectional relationships.

Such bidirectional relationships may be common but are not strictly required in TSP.

Endpoints **A** and **B** can establish multiple relationships using distinct and statistically unrelated VIDs. For instance, if endpoint **A** manages a set of VIDs represented as VID_a_0, VID_a_1, VID_a_2, and endpoint **B** manages a set labeled VID_b_0, VID_b_1, VID_b_2, then they can create various relationships like:

In endpoint **A**: (VID_a_0, VID_b_0), (VID_a_1, VID_b_1), (VID_a_2, VID_b_2)
In endpoint **B**: (VID_b_0, VID_a_0), (VID_b_1, VID_a_1), (VID_b_2, VID_a_2).

The VIDs and the relationships they appear in an endpoint define the *contexts* in which they are used. The endpoints must maintain such contexts throughout a VID's and its associated relationships lifecycle. In other words, context must be persisted. The context database must be properly secured by the endpoints to minimize the risk of exposure.

3.7 Message Header, Body and Plaintext Signature

The TSP envelope structure offers two locations for a message by the sender's choice. Regardless of which location, the message itself has the same format including a Message Header, a Message Body and an optional Message Signature field. As noted earlier, this Signature field applies to the message in plaintext and may be necessary for some applications because of legal or other requirements. For a receiver of a message, it interprets the data enclosed in the message in the same way other than, obviously, the fact that they will have different trust properties. To avoid confusion, we always refer to the fields in the envelope itself as the Envelope to distinguish them from the Message.

3.7.1 Message Header

```
Message_Header = {
      Type,      //Required
      Subtype,   //Required
      …
      …
}
```

The `Type` field is a numerical code exclusively allocated in this specification. The `Subtype` field is a numerical code allocated within a given `Type` code's scope. Such `Subcode` can be allocated by another specification that defines the specific meanings of the associated `Type` code. Typically, one or more `Type` codes can be used by a higher layer protocol that utilizes TSP messages.

3.7.2 Message Body

```
Message_Body = {
      …
      …
}
```

3.7.3 Message Signature

Optional
Only for ETS Messages since S Messages are already in plaintext.

3.7.4 General Message

```
Type = TSP_GEN_MSG
Subtype = NA
```

These are generic messages.

3.7.5 Hello Message

```
Type = TSP_CTL_MSG
Subtype = HELLO
```

A HELLO message is used by an endpoint when it knows a destination `VID_rcvr` and wishes to establish a relationship with one of its own `VID_sndr`: `<VID_sndr, VID_rcvr>`. The HELLO message is the first message in this context.

3.8 Sender Procedures

We outline the procedures for TSP message senders in two parts: the initial message which establishes the relationship, and the follow-up messages that occur within that established relationship.

Endpoint **A**, which controls `VID_a` associated with Support System **A***, acquires `VID_b` of Endpoint **B** through an out-of-band introduction (OOBI) method. `VID_b` is tied to Support System **B***. Note that **A*** could be the same as or different from **B***. If Endpoint **A** selects to employ `VID_a` to dispatch a TSP message to the Endpoint identified by `VID_b` for the first time, it will be establishing a unidirectional relationship denoted by `<VID_a, VID_b>`.

Here is the procedure Endpoint **A** follows when sending its inaugural message to `VID_b`:

Step 1: Resolve `VID_b` to acquire access to the following mandatory information

- Appraisal information as specified in Section 2.1.5
- Public key
- VID verification information
- Transport information

Step 2: Verify and Appraise
Step 3: Create TSP message, for example, `{VID_sndr, VID_rcvr, Ciphertext,Signature}`
Step 4: Use the retrieved transport access point to send the message.
Step 5: Update relationship table with `<VID_a, VID_b>`.

For subsequent messages, the procedure is simpler:

Step 1: Create TSP message, for example, `{VID_sndr, VID_rcvr, Ciphertext,Signature}`
Step 2: Use the retrieved transport access point to send the message.

3.9 Receiver Procedures

Endpoint **B** receives a TSP message: `{VID_x, VID_y, Ciphertext, Signature}`. Endpoint **B** uses the following procedure to process this incoming message:

Step 1: Check if `<VID_y, VID_x>` matches an existing relationship. If yes, say it matches `<VID_b, VID_x>`, then jump to Step 5; otherwise this is the first message of this relationship.
Step 2: Check if `VID_y` is a valid local VID and local rules permit to proceed.
Step 3: Resolve `VID_x` to acquire access to the following mandatory information

- Appraisal information as specified in Section 2.1.5
- Public key
- VID verification information
- Transport information

Step 4: Verify and Appraise
Step 5: Verify the Signature with `VID_x.pk`.
Step 6: Decrypt the Ciphertext with `VID_y.sk`. Verify that `VID_sndr` in the ETS envelope matches a VID of a remote entity in the relationship table and matches the S Envelope's sender VID.
Step 7: Process the message.

3.10 Handling Changes

3.11 Out of Band Introductions

Before an endpoint A can send the first TSP message to another endpoint B, it must somehow discover at least one VID that belongs to B. If they also wish to utilize the routed mode, as specified in Section 5, then additional VIDs may also be needed before the first TSP routed message can be sent. We call any such method that could help the endpoints discover such prerequisite information an Out of Band Introduction (OOBI). There may be many such OOBI methods. Detailed specifications of OOBI methods are out of scope.

For the purpose of TSP, information obtained from OOBI methods must not be assumed authentic, confidential or private, although mechanisms to remedy such vulnerabilities should be adopted whenever possible. TSP implementations must handle all cases where the OOBI information is not what it appears. Please refer to Section 3.8, 3.9, and Section 5 for the details of what OOBI information is required and how it should be used.

4. Nested Messages

When TSP sender '***a***' dispatches an Authentic Confidential Message intended for receiver '***b***', the observable data structure for any third party entity not involved in the message exchange between ***a*** and ***b*** appears as:

```
{VID_a, VID_b, Ciphertext, Signature}
```

Over time, with a sustained exchange of such messages, an external observer may accumulate a significant volume of data. This data, once analyzed, could reveal patterns related to time, frequency, and size of the messages. Using `VID_a` and `VID_b` as keys, an observer can index this dataset. It's then possible to correlate this indexed data with other available metadata, potentially revealing more insights into the communication.

To mitigate this threat, TSP offers a technique whereby parties encapsulate a specific conversation — for instance, a sequence of messages — within an additional TSP envelope, as described below.

4.1 Nested Envelope and Contextual Relationship

Suppose endpoints ***a*** and ***b*** opt for VIDs, namely $\texttt{VID\_a}_1$ and $\texttt{VID\_b}_1$, for a specific conversation. There might be concerns that these VIDs could be linked to other external metadata, potentially exposing the conversation to unwanted scrutiny. To counteract this risk, the two endpoints can use an alternative pair of VIDs - $\texttt{VID\_a}_0$ and $\texttt{VID\_b}_0$. These alternative VIDs should be chosen such that they are less susceptible to linkage with other available metadata. With these other pair of VIDs and the established relationships $(\texttt{VID\_a}_0, \texttt{VID\_b}_0)$, they can then nest their communication inside an additional envelope in the following manner:

$$
\begin{aligned}
&\{\texttt{VID\_a}_0, \texttt{VID\_b}_0, \texttt{Ciphertext}_0(\texttt{VID\_a}_0,\\
&\quad \{\texttt{VID\_a}_1, \texttt{VID\_b}_1, \texttt{Ciphertext}_1(\texttt{VID\_a}_1, \texttt{Message}), \texttt{Signature}_1\}),\\
&\quad \texttt{Signature}_0\}
\end{aligned}
$$

Where,

$$\texttt{Ciphertext}_1 = \texttt{Encrypt}_{\texttt{VID\_b1.PK}}(\texttt{VID\_a}_1, \texttt{Message}),$$

$$\texttt{Signature}_1 = \texttt{Sign}_{\texttt{VID\_a1.SK}}(\texttt{VID\_a}_1, \texttt{VID\_b}_1, \texttt{Ciphertext}_1),$$

$$\texttt{Ciphertext}_0 = \texttt{Encrypt}_{\texttt{VID\_b0.PK}}(\texttt{VID\_a}_0, \texttt{VID\_a}_1, \texttt{VID\_b}_1, \texttt{Ciphertext}_1, \texttt{Signature}_1),$$

$$\texttt{Signature}_0 = \texttt{Sign}_{\texttt{VID\_a0.SK}}(\texttt{VID\_a}_0, \texttt{VID\_b}_0, \texttt{Ciphertext}_0).$$

TBS: do we leave compacting optional or always use compacting as defined in Section 4.2?

Note that the above example uses Authentic Confidential Message (ACM) envelope structure, the nesting procedure however can be applied to all envelope structures. Furthermore, the inner envelope and the outer envelope in a nested message may use different envelope structures.

To a non-participant, the observed pattern in this scheme will be:

$\{VID\_a_0, VID\_b_0, Ciphertext_0, Signature_0\}.$

While the observer might gather data linked to (VID_a_0, VID_b_0), they won't be able to know about VID_a_1 or VID_b_1 or link them with the respective endpoints, and, won't be able to distinguish the nested messages associated with (VID_a_1, VID_b_1) from the overall set of messages protected by the outer envelope.

When TSP messages utilize this nesting approach, the resulting relationship, (VID_a_1, VID_b_1), is termed a "*context*." The privacy protection afforded by this method is designated as one example of "*contextual privacy*." Since the nested messages hide the inner VID pair from being collected as a part of potential correlation attacks, we also refer to this style of privacy protection as "*correlation privacy*."

The process for establishing such contextual relationships is detailed in Section 5.2. It's important to note that this nesting can be recursively applied, adding additional layers as required. Contextual relationships are situated within a foundational relationship that has been vetted (verified and appraised) and deemed suitable for the intended purpose by both participating endpoints. The VIDs engaged in these contextual relationships are exempt from undergoing their own verification and appraisal procedures and do not necessitate resolution to specific transport mechanisms.

4.2 Compact Nested Envelope

The direct TSP message uses ESSR to ensure authenticity and confidentiality. When a nested message is used to enhance contextual privacy between VIDs of the same pair of endpoints as the outer layer, it may be unnecessary to encrypt or sign the inner message again. In such scenarios, the nested message can be compacted as follows:

$$\{VID\_a_0, VID\_b_0, Ciphertext_0(VID\_a_0, \{VID\_a_1, VID\_b_1, Message\}), Signature_0\}$$

Where,

$Ciphertext_0 = Encrypt_{VID\_b0.PK}(VID\_a_0, VID\_a_1, VID\_b_1, Message)$,

$Signature_0 = Sign_{VID\_a0.SK}(VID\_a_0, VID\_b_0, Ciphertext_0)$.

The sender may choose to also sign the inner message as follows:

$$\{VID\_a_0, VID\_b_0, Ciphertext_0(VID\_a_0,$$

$\{VID\_a_1, VID\_b_1, Message, Signature_1\}), Signature_0\}$

Where,
$Ciphertext_0 = Encrypt_{VID\_b0.PK}(VID\_a_0, VID\_a_1, VID\_b_1, Message, Signature_1)$,
$Signature_1 = Sign_{VID\_a1.SK}(VID\_a_1, VID\_b_1, Message)$,
$Signature_0 = Sign_{VID\_a0.SK}(VID\_a_0, VID\_b_0, Ciphertext_0)$.

As further detailed in Section 9, when CESR is used for message encoding, its self-framing feature allows endpoints to choose the above options without extra codes in the envelope headers.

4.3 Shorthand Notation for Nested Envelope

For clarity and ease of representation, we adopt the shorthand notation **[...]** first introduced in Section 3 to depict a message encapsulated within an envelope when the details of the envelope is not the main concern:

[VID_a_0, VID_b_0, Msg]

Here, VID_a_0 denotes the sender, and VID_b_0 signifies the receiver. The Message, Ciphertext and Signature are implicitly assumed to align with the requirements for proper TSP envelope formats. For Authentic Confidential Message envelope, the shorthand notation can be precisely defined as follows:
[VID_a_0, VID_b_0, Msg] = $\{ VID\_a_0, VID\_b_0, Ciphertext, Signature \}$,
Where,
$Ciphertext = Encrypt_{VID\_b0.PK} (VID\_a_0, Msg)$,
$Signature = Sign_{VID\_a0.SK} (VID\_a_0, VID\_b_0, Ciphertext)$.

A message nested within another can then be succinctly expressed as:

[VID_a_0, VID_b_0 [VID_a_1, VID_b_1, Msg]]

Again, for Authentic Confidential Message envelope, the above nested shorthand can be precisely defined as follows:
[VID_a_0, VID_b_0 [VID_a_1, VID_b_1, Msg]] =
$\{ VID\_a_0, VID\_b_0, Ciphertext_0, Signature_0 \}$,
$Ciphertext_0 = Encrypt_{VID\_b0.PK} (VID\_a_0, Msg_0)$,
$Signature_0 = Sign_{VID\_a0.SK} (VID\_a_0, VID\_b_0, Ciphertext_0)$,
$Msg_0 = \{ VID\_a_1, VID\_b_1, Ciphertext_1, Signature_1 \}$,
$Ciphertext_1 = Encrypt_{VID\_b1.PK} (VID\_a_1, Msg)$,
$Signature_1 = Sign_{VID\_a1.SK} (VID\_a_1, VID\_b_1, Ciphertext_1)$.

We do have to be careful that the two **[...]** in this nesting may actually have different structures when using such a notation. For simplicity and not to overburden the notation, the ACM envelope is used in nested messages even though other envelopes could also be used as in Sections 6 and 7.

5. Routed Messages Through Intermediaries

Intermediaries are systems utilized by endpoints to enhance various aspects of communication, such as asynchronous delivery, reliability, performance, among others. In this specification, our primary focus is on their role in ensuring contextual privacy protection for communications between endpoints.

5.1 Contextual Privacy in Routed Mode

Contextual privacy is one of the primary goals of deploying TSP in the routed mode. The TSP endpoints, the sender and receiver, aim to route their messages through chosen intermediaries, maintain the same authenticity and confidentiality properties of TSP and enhance the protection of contextual privacy related to the following exposures:

- The exposed direct neighbor relationship VIDs and related network transport information used to carry TSP messages are publicly knowable by all third parties. The TSP routed mode shields exposure of VIDs in endpoint-to-endpoint contextual relationships through nested envelopes as defined in Section 4.
- VIDs used in routing and part of route information are knowable by the intermediaries along the routing path by necessity. The intermediaries are given only limited trust related to carrying out routing functions. Another layer of nesting allows endpoints to shield their contextual relationship VIDs from the intermediaries in the routing path.

In the high level, an overall endpoint-to-endpoint TSP routed mode involves three types of relationships.

- Direct neighbor relationships
    - Sender and its intermediary relationship
    - Intermediary to intermediary relationship
    - Receiver and its intermediary relationship
- Endpoint-to-endpoint relationship
- Nested private endpoint-to-endpoint relationship

The overall TSP routing is accomplished with a combination of designating intermediaries in the routing path and unwrapping nested messages and routing via direct neighbor relationship. The neighbors may create a routing context specific relationship for the purpose of routing en route messages. A typical three hop pattern of TSP routing is illustrated in Figure 5 which we will reference to define procedures for endpoints (source and destination) and intermediaries. In Figure 5, the outer envelope is for routing messages through direct neighbor relationships, the middle envelope is for endpoint-to-endpoint messages, and the orange envelope is for nested

private endpoint-to-endpoint messages. We will describe the details of the routed mode operations step by step in the following Sections 6.2 to 6.7.

Following this basic model, we then define a generalized routing model in Section 6.8.

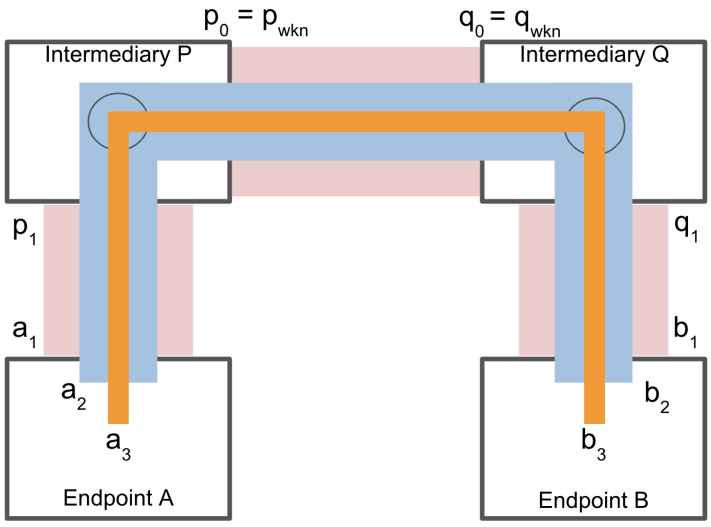

Figure 5: Routed Mode in Three Hops

5.2 Routed Message Envelope

For routed messages, we need to distinguish the terms “sender” and “source”, and “receiver” and “destination”. We reserve the terms “sender” and “receiver” for direct neighbors relationships between who the message is being transported from one address to another (i.e. being routed). We reserve the terms “source” and “destination” for endpoint-to-endpoint relationships between whom the carried inner message is being communicated.

The TSP routed message envelope does not change the S Envelope but extends the ETS Envelope with the following structure:

```
S_Envelope = {
     VID_sndr,        // Direct neighbor
     VID_rcvr,        // Direct neighbor
}

ETS_Envelope = {
     VID_sndr,        // Direct neighbor
     VID_nexthop,     // Destination’s intermediary public VID
     VID_exit,        // Destination’s intermediary private VID for
the destination
}
```

In the ETS Envelope, the VIDs following the first VID_sndr may be considered as an ordered list of next hop VIDs by the intermediary receiver. The list can vary and should be interpreted as in the order of a routing path with the second VID coming first.

In our shorthand notation, we also include the destination's intermediary VIDs.

[VID_sndr, VID_rcvr, VID_nexthop, VID_exit, Msg]

denotes that `VID_sndr` and `VID_rcvr` are found in the S envelope (signed but not encrypted), whereas `VID_sndr`, `VID_nexthop`, and `VID_exit` are found in the ETS envelope (encrypted then signed).

The intermediary (i.e. the entity processing this message) decrypts and sees `VID_nexthop` as the next hop VID, then attempts to route the message to the party identified by `VID_nexthop`.

The above shorthand notation can be defined as follows for an ACM envelope:

[VID_sndr, VID_rcvr, VID_nexthop, VID_exit, Msg] =
{VID_sndr, VID_rcvr, Ciphertext, Signature},
Ciphertext = $\text{Encrypt}_{\text{VID\_rcvr.PK}}$({VID_sndr, VID_nexthop, VID_exit, Msg}),
Signature = $\text{Sign}_{\text{VID\_sndr.SK}}$(VID_sndr, VID_rcvr, Ciphertext).

Note that for any third party, this message appears as a normal TSP message in the form of:
{VID_sndr, VID_rcvr, Ciphertext, Signature}.

5.3 Direct Neighbor Relationship and Routing

Endpoint **A** chooses an intermediary, denoted as **P**, and forms a bidirectional neighbor relationship. In Figure 6, the neighbor relationship between **A** and **P** is illustrated as: $(\text{VID\_a}_1, \text{VID\_p}_1)$ which is assumed to be established before message routing takes place. This assumption also applies to neighbor relationships between intermediaries, and between endpoint **B** and its intermediary, as shown in Figure 6. Message routing between endpoint **A** and endpoint **B** takes place within this established network of relationships.

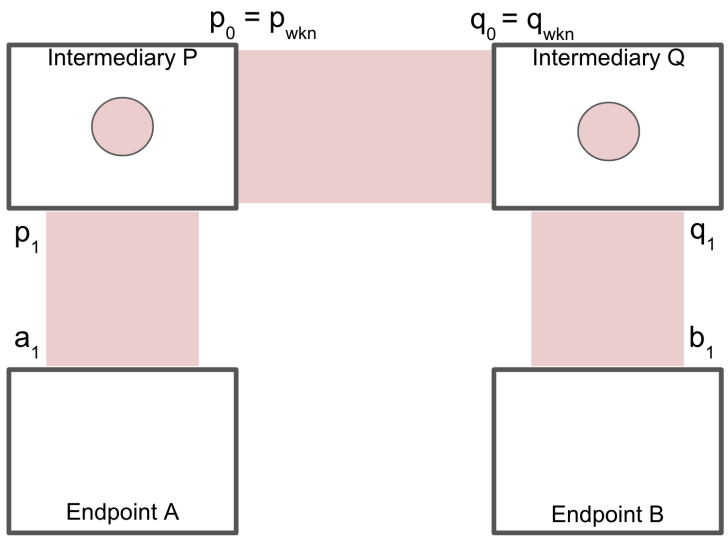

Figure 6: Direct neighbor relationships

These direct neighbor relationships allow normal TSP messages such as:

- **[VID_a_1, VID_p_1, Msg]**
- **[VID_p_0, VID_q_0, Msg]**
- **[VID_q_1, VID_b_1, Msg]**

They can also be used to route messages when the receiver is an intermediary. We will detail each party's operations in the following sections.

5.3.1 The Source Endpoint

The source endpoint **A** can use the envelope defined in Section 6.2 to send the following routed message to intermediary **P**:

- **[VID_a_1, VID_p_1, VID_q_0, VID_q_1, Msg]**

The routing VIDs (VID_q_0 and VID_q_1) become known to endpoint A prior to this via an OOBI or other discovery protocol that is out of scope. Note that in this outer layer, all VIDs shown in Figure 6 are public while p_0 and q_0, as public VIDs of intermediaries, may also be well known. If p_0 and q_0 are made to be well known, they are shown as p_{wkn} and q_{wkn} in Figure 6. Intermediaries **P** and **Q** are, of course, not required to do so.

5.3.2 The Source's Intermediary

The source's intermediary **P** must support routed messages. In the message received (shown in the preceding section), **P** processes the first two VIDs as normal sender and receiver VIDs normally. The next VID in the list, VID_q_0, is the next hop's VID. **P** must attempt to route the carried message to the next hop.

If (VID_p_0, VID_q_0) relationship is pre-existing, **P** knows how to forward the message. If it is not pre-existing but VID_q_0 is well known, **P** can resolve it and establish a new (VID_p_0, VID_q_0) relationship using normal procedures specified in Section 3. **P** then route the message to **Q** using the following envelope:

- **[VID_p_0, VID_q_0, VID_q_1, Msg]**

5.3.3 The Destination's Intermediary

The destination's intermediary, Q, also processes the first two VIDs as normal sender and receiver VIDs normally. The next VID in the list, VID_q_1, is the next hop's VID. **Q** must attempt to route the carried message to the next hop.

Since VID_q_1 is given to endpoint **A** by **B** itself, the (VID_q_1, VID_b_1) relationship should be pre-existing, and **Q** knows how to forward the message. If it is not pre-existing, this is an error (See Section on error handling TBD). **Q** then route the message to **B** using the following envelope:

- **[VID_q_1, VID_b_1, Msg]**

5.3.4 The Destination

When the destination receives the message it no longer has any next hop VID. Note that endpoints are not required to handle routed messages that contain additional next hop VID or VIDs.
Unlike direct mode messages, this message's sender VID_q_1 is of the intermediary **Q**, but the source **A**; and its receiver VID_b_1 is associated with the relationship with **Q**, not **A**. It means that the destination endpoint **B** can not be assured of the message's authenticity, confidentiality not contextual privacy. To solve these problems, we specify additional two procedures in the following sections.

5.4 Endpoint-to-Endpoint Messages

In Section 6.3, we defined a routed operation method that enables a source endpoint to send a TSP message to a destination endpoint via a series of intermediaries, using a hop-by-hop approach. However, while this approach ensures successful message delivery from the source to the destination, it doesn't uphold the core trust properties TSP aims to provide—specifically, authenticity, confidentiality, and contextual privacy— in the context of third parties or intermediaries. In this section, we define the endpoint-to-endpoint message envelope carried within the payload of routed messages and the corresponding endpoint-to-endpoint relationship which ensures authenticity, confidentiality, and a degree of contextual privacy. This operation is illustrated in Figure 7 below.

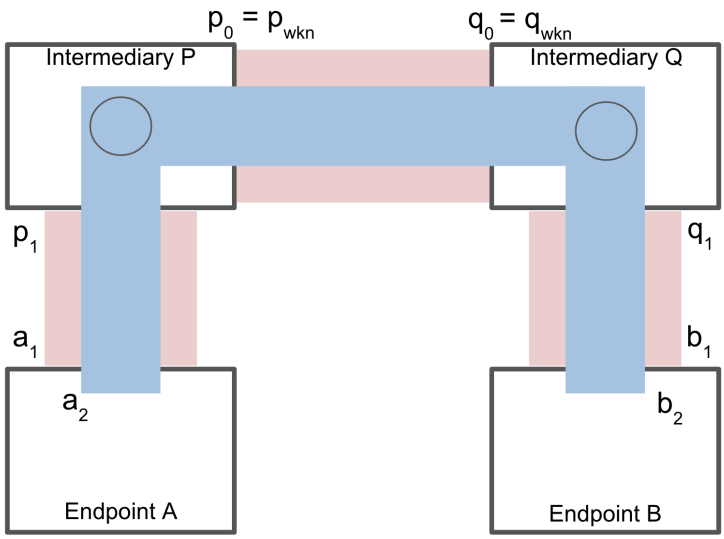

Figure 7: Endpoint-to-Endpoint relationship between endpoints **A** and **B** through a routed path

5.4.1 The Source Endpoint

The source endpoint **A** will create an endpoint-to-endpoint relationship with endpoint **B** using the same procedure specified in Section 3. Instead of direct messages in Section 3, the endpoint **A** will use routed messages defined in Section 6.3. Recall in Section 6.3.1, endpoint A sends the following message to intermediary **P** en route to eventual destination **B**:

- **[VID_a_1, VID_p_1, VID_q_0, VID_q_1, Msg]**

Endpoint A will encapsulate its relationship forming direct message with endpoint B as follows:

- **[VID_a_1, VID_p_1, VID_q_0, VID_q_1, [VID_a_2, VID_b_2, Msg_e]]**

Because this is the first layer where endpoint-to-endpoint communication takes place, the source must use its own encryption and signing and not opt out as described in Section 4.2.

5.4.2 The Destination Endpoint

As described in Section 6.3, this message will be delivered to the destination **B** in the form of,

- **[VID_q_1, VID_b_1, Msg]**

The inner message is

- **Msg = [VID_a_2, VID_b_2, Msg_e]**

It is routed transparently by the intermediaries. Note that the intermediaries do have visibility to VID_a_2 and VID_b_2 but not Msg_e which is in the ETS Message body. By unwrapping the outer message from intermediary **Q**, within relationship (VID_b_1, VID_q_1), the destination **B** receives,

- **[VID_a_2, VID_b_2, Msg_e]**

Destination **B** now has a direct message from the source with VID_a_2 and addressed to its own VID_b_2 and can perform the same procedure as specified in Section 3 to ensure authenticity and confidentiality, and establish relationship <VID_a_2, VID_b_2>. In terms of contextual privacy, VID_a_2 and VID_b_2 are not visible to third parties but are visible to intermediaries **P** and **Q**.

Intermediaries should not process VID_a_2 and VID_b_2 and must not store VID_a_2 and VID_b_2 in any persistent storage in order to minimize the risk of exposure of these private endpoint-to-endpoint VIDs.

As described in Section 4, endpoints may use nested messages to further strengthen contextual privacy. In the next section, we specify such a nested envelope such that contextual VIDs between endpoints **A** and **B** can be hidden from the intermediaries as well.

5.5 Nested Private Endpoint-to-Endpoint Messages

In this section, we specify an operation using nested messages over the endpoint-to-endpoint messages described in the previous section. The purpose of this nested envelope is to hide the private contextual VIDs from being visible by the intermediaries.

The nested private endpoint-to-endpoint pattern is illustrated in Figure 5 and replicated in Figure 8 for convenience of reference.

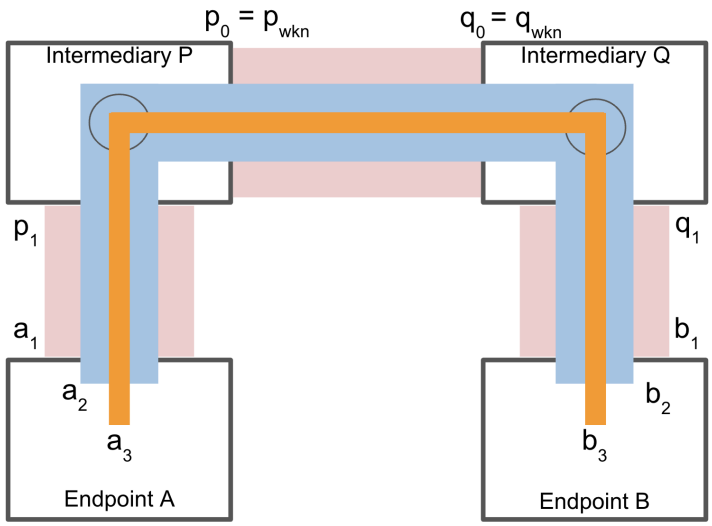

Figure 8: Nested Private Endpoint-to-Endpoint Messages

5.5.1 The Source Endpoint

Using procedures defined in Section 4 and 5, endpoints A and B choose VID_a_3 and VID_b_3 respectively for the private contextual relationships. The source A then sends its message to B over the envelope described in the previous section as follows:

- **[VID_a_1, VID_p_1, VID_q_0, VID_q_1, [VID_a_2, VID_b_2, Msg_e]]**

The third level of encapsulation is,

- **Msg_e = [VID_a_3, VID_b_3, Msg_{pe}]**

Since Msg_e is inside of the endpoint-to-endpoint encrypted payload, VID_a_3 and VID_b_3 are not visible to intermediaries.

5.5.2 The Destination Endpoint

As described in Section 6.4, the destination B receives:

- **[VID_q_1, VID_b_1, [VID_a_2, VID_b_2, Msg_e]]**
- **Msg_e = [VID_a_3, VID_b_3, Msg_{pe}]**

It then decrypts the endpoint-to-endpoint encrypted Msg_e, and then verifies and decrypts with the more private relationship <VID_a_3, VID_b_3> and receives the ultimate message Msg_{pe}.

5.6 Using The Same Intermediary

The endpoints **A** and **B** may use the same intermediary, i.e, **P** = **Q**. Since **A** and **B** usually choose their intermediaries independently, this scenario may happen by coincidence. Regardless of how it occurs, the operation specified in this section continues to ensure the same trust properties except the fact that compromise of a single intermediary may expose the whole routing path.

5.7 Routing With More Than Two Intermediaries

In the previous sections, we outlined a three-hop pattern involving two intermediaries for routing TSP messages and examined the special case where two endpoints use the same intermediary. In this section, we expand on this pattern to incorporate scenarios involving more than two intermediaries.

The routed message envelope specified in Section 6.2 can be extended easily as follows to allow more than two intermediaries.

```
S_Envelope = {
     VID_sndr,        // Direct neighbor
     VID_rcvr,        // Direct neighbor
}

ETS_Envelope = {
     VID_sndr,        // Direct neighbor
     VID_nexthop1,    // The first next hop intermediary's public VID
     …
     VID_nexthopk,    // The kth next hop intermediary's public VID
     VID_exit,        // Destination's intermediary private VID for
the destination
}
```

The shorthand notation is also extended to include more next hop intermediaries.

[VID_sndr, VID_rcvr, VID_nexthop$_1$, …, VID_nexthop$_k$, VID_exit, Msg] =
{VID_sndr, VID_rcvr, Ciphertext, Signature},
Ciphertext = Encrypt$_{VID_rcvr.PK}$({VID_sndr, VID_nexthop$_1$, …,
VID_nexthop$_k$, VID_exit, Msg}),
Signature = Sign$_{VID_sndr.SK}$(VID_sndr, VID_rcvr, Ciphertext).

The ordered list VID_nexthop$_1$, ..., VID_nexthop$_k$, specifies the routing path for a TSP routed message. As the message traverses this path, each intermediary in the sequence decrypts the ETS Envelope and retrieves the foremost next hop VID from the list. It then proceeds to forward the message, carrying along the reconstituted VID_sndr and any remaining VIDs in the next hop list, the VID_exit in the new ETS Envelope and the message payload.

When the message reaches the final intermediary in the path, this intermediary is responsible for forwarding the message to its destination. This is done using a regular TSP envelope, directed to the destination as determined by the neighbor relationship associated with VID_exit. This process ensures that the message systematically and securely progresses through the specified intermediaries, ultimately reaching its intended endpoint.

It's important to note that in extending the routing layer for more complex paths, only the routed layer itself requires modification. This involves replacing the singular next hop VID, as initially defined in Section 6.3, with a simple ordered list to represent the routing path. The endpoint-to-endpoint relationships over the routed path remain unaffected by these routing layer changes. They are transparent to the routing process and do not necessitate any alterations, ensuring that the fundamental structure of endpoint-to-endpoint communication can be preserved should we adopt additional routing methods in the future.

6. Multi-Recipient Communication Methods

When an endpoint aims to send an identical message to multiple recipient endpoints, it engages in a communication pattern that could be labeled as multicast, broadcast, or anycast. However, the definitions and semantics of these terms can be complex, often ambiguous, and vary significantly depending on the context. Therefore, in this specification, we choose not to adopt these specific terms.

Instead, each method outlined in this section is defined with its own clear understanding of what it means to send the same message to multiple endpoints. Moreover, these methods are explicit about the precise trust properties they guarantee during such multi-recipient communication.

It's worth noting that additional methods, catering to such multi-recipient messaging while ensuring specific trust properties, may be detailed in separate future specifications. This approach allows for clarity and specificity in defining how TSP handles messages intended for multiple endpoints, without the ambiguity often associated with traditional multicasting terminologies.

6.1 Source's Receiver List

A source endpoint A may send a message to multiple recipients by simply maintaining a set of recipient VIDs associated with a single VID_a and repeating the same message to each recipient in the set.

<VID_a, VID_b$_1$>,
<VID_a, VID_b$_2$>,
…
<VID_a, VID_b$_k$>

TODO

6.2 Unaddressed Messages

A common pattern of communication is for a source endpoint to use an authentic message envelope but not addressed to a specific recipient. Such messages are therefore intended to be authenticable but non-confidential. The envelope previously defined in Section 3.5.2 can be used for this purpose.

TODO

7. Control Messages

This section outlines control messages that are either necessary or beneficial for the proper functioning of the Trust Spanning Protocol, referred to here as control messages. All such control messages will utilize the envelope formats as specified within this document.

For control messages aiding common two-party relationships, the Authentic Confidential Message (ACM) envelope defined in Section 4.4 is employed:

In direct mode: {VID_sndr, VID_rcvr, Msg}
In routed mode: {VID_sndr, VID_rcvr, VID_nexthop_list..., Msg}
Additionally, Section 6.2 discusses unaddressed messages and delves into that particular scenario.

The fields within the envelope are encoded using CESR, but the higher layer payload may use JSON, CBOR or MGPK serializations, including interleaving, as supported by CESR. The Msg data is structured as follows: {Type, Subtype, Payload}, where Type is always TSP_CTL for control messages, and Subtype varies depending on each specific control message defined in this section.

The Payload section is designed to be extendable. While we define the necessary TSP-control-fields (both mandatory and optional), higher layers have the flexibility to expand upon these:

Payload = {TSP-control-fields, Extended-fields}

This structure ensures a standardized approach for the essential components of the message while allowing adaptability for specific use cases or additional requirements at the higher layer. Also note that control messages may carry additional user payload in addition to the control fields we specify here. This is useful to reduce round trip delays otherwise would be unnecessarily imposed on applications.

7.1 Relationship Forming

7.1.1 Direct Relationship

When an endpoint A learns from another endpoint B the VID for B, say VID_b, through an OOBI method, the endpoint A may use the following message type to form a direct relationship with B. Suppose the source VID that endpoint A uses is VID_a, then the relationship A and B establishes is (VID_a, VID_b).

OOBI: VID_b
TSP Envelope: Authenticated Confidential Message (ACM) = [VID_a, VID_b, Msg]

Msg Payload:

Type = TSP_CTL
Subtype = NEW_REL
TSP-control-fields = Nonse

Endpoint B retrieves and verifies VID_a, and if agrees, replies with the following:
Envelope: [VID_b, VID_a, Msg]
Msg Payload:
Type = TSP_CTL
Subtype = NEW_REL_REPLY
TSP-control-fields = Thread-ID = Digest(B's received message)

The result is a bi-directional relationship (VID_a, VID_b) in both endpoints. The Thread-ID is recorded by both endpoints and used in all future messages.

If endpoint B

- fails to verify VID_a,

it SHOULD direct the transport layer to disconnect or otherwise block or filter out further incoming messages.

If endpoint B

- for any other reason, does not want to or can not engage with endpoint A,

it MAY simply remain silent (if B does not want to give A any private information), or it MAY reply with a REL_CANCEL message as specified in Section 7.4 with proper event code (if B is willing to risk additional information disclosure by providing A some useful information).

If endpoint B

- is OK with receiving the incoming messages from endpoint A, but declines to reply to endpoint A to establish the opposite direction relationship,

it MAY simply remain silent.

Other actions that endpoint B may take are left unspecified.

In all of the above cases, the responding party (endpoint B) should be careful about privacy leaks if it chooses to respond to an incoming message. The more private option is to remain silent.

7.1.2 Relationship over a Routed Path

When an endpoint A learns from another endpoint B through an OOBI method the VID for B, say VID_b, together with a routed path, say [VID_b, VID_nexthop$_1$, ..., VID_nexthop$_k$, VID_exit], endpoint A may use the following message type to form a relationship with B. Suppose the source VID that endpoint A uses is VID_a, and optionally endpoint A specifies a return routed path VID_rethop$_1$, ..., VID_rethop$_k$, VID_retexit, then the relationship A and B establishes is (VID_a, VID_b).

OOBI: VID_b, VID_nexthop$_1$, ..., VID_nexthop$_k$, VID_exit
Envelope: Authenticated Confidential Message (ACM) = [VID_a, VID_b, VID_nexthop$_1$, ..., VID_nexthop$_k$, VID_exit, Msg]

Msg Payload:
Type = TSP_CTL
Subtype = NEW_REL
TSP-control-fields = {Nonce, VID_rethop$_1$, ..., VID_rethop$_k$, VID_retexit}

Endpoint B retrieves and verifies VID_a, and if agrees, replies with the following:
Envelope: [VID_b, VID_a, VID_rethop$_1$, ..., VID_rethop$_k$, VID_retexit, Msg]
Msg Payload:
Type = TSP_CTL
Subtype = NEW_REL_REPLY
TSP-control-fields = Thread-ID = Digest(B's received message)

In this scenario, either A or B may choose not to specify a routed path for its incoming message. If one party specifies a routed path while the other party does not (but they both agree to such an arrangement), then the result can be a relationship where it is over a routed path in one direction but direct in the other direction.

The result is a bi-directional relationship (VID_a, VID_b) in both endpoints. The Thread-ID is recorded by both endpoints and used in all future messages.

TODO: exceptions

7.2 Parallel Relationship Forming

If endpoints A and B have a relationship (VID_a0, VID_b0), they can establish a new parallel relationship using the current relationship as a way of referral.

Endpoint B sends to A: [VID_b0, VID_a0, ..., Msg] (The '...' denotes the omitted nexthop list for routed mode)
Subtype=NEW_REFER_REL
Payload = {VID_b1, NULL | Nexthop-List}

When endpoint A receives this message from B and it treats it as an introduction, then A initiates a normal new relationship forming procedure as specified in Section 7.1.

In this procedure, VID_b1 is the new VID for endpoint B. If endpoint A picks VID_a, then the new relationship (VID_a1, VID_b1) is parallel to (VID_a0, VID_b0).

If 'nexthop-list' is present, then A uses the specified routed path to send the NEW_REL message to endpoint B.

7.3 Nested Relationship Forming

If endpoints A and B have a relationship (VID_a0, VID_b0), they can also establish a new nested relationship using the current relationship. The new relationship can be private as discussed in Section 2.1.

Endpoint A sends to B: [VID_a0, VID_b0, ..., [VID_a1, NULL, Msg]]
Subtype=NEW_NEST_REL
Payload = VID_a1_{pk} | ResVeriInfo

ResVeriInfo is a field defined by the VID as info for VID resolution and verification, e.g. KERI Key Event Log. The detail format of this field is to be specified by individual VID specifications.

TBD: add an example for did:webs and AID.

Endpoint B replies to A: [VID_b0, VID_a0, ..., [VID_b1, VID_a1, Msg]]
Subtype=NEW_NEST_REL_REPLY
Payload = {VID_b1_{pk} | ResVeriInfo, Thread-ID}

The new relationship formed by the above control message exchange is: (VID_a1, VID_b1). Because the relationship (VID_a1, VID_ b1) is private, the verification is done through the above two messages privately. No address resolution procedure is supported.

The current relationship can be direct or over routed mode, the same procedure applies. Similarly, the current relationship itself can be a nested relationship, the same procedure applies. The resulting relationship (VID_a1, VID_b1) can only be used in a nested message.

7.4 Relationship Events

7.4.1 Key Rotation Update

Key rotation is a beneficial feature that may be supported by some VID types. Applications may use such a key rotation method to enhance their longer term trust properties. While key rotation mechanisms are mostly a part

of the VID support system, the key rotation events impact the operation of the endpoints engaged in a long term TSP relationship.

When a TSP sender notices a key rotation of its key pairs bound with VID_sndr, it SHOULD set a KR flag in the message header of the first TSP message. This flag serves as an informational indicator for the receiver to initiate the re-verification process and begins to switch keys. The receiver, if successfully re-verify the VID_sndr, updates its VID record of the new sender's public key. The flag MAY be repeated in more than one message after a key rotation event.

7.4.2 Route Info

TODO
This section is to provide an informational message from an Intermediary to either another Intermediary or an endpoint to provide route information, for instance, "UNABLE TO REACH", that could be useful for legitimate diagnoses without introducing contextual privacy risks. Details TODO.

7.4.3 Relationship Cancellation

Bidirectional relationships in TSP are essentially a combination of two unidirectional relationships that involve the same pair of VIDs. Due to the asymmetric nature of TSP messages, it's possible for a relationship to exist unilaterally for a time—where messages flow in one direction but not yet in the reverse. This scenario can occur both when a relationship is being established and when it's being terminated.

While sending explicit messages to cancel a relationship is not an absolute necessity within TSP, such messages can be beneficial for upper-layer protocols that require a clear and definite termination of relationships. For this purpose, endpoints utilize REL_CANCEL messages.

The process for canceling a relationship is uniform, regardless of whether the relationship uses a direct transport or a routed transport path. This consistency ensures that the cancellation of relationships is handled systematically and efficiently, aligning with the overarching protocol design for managing endpoint relationships.

For a relationship denoted as (VID_a, VID_b), either endpoint A or B can initiate the cancellation by sending a REL_CANCEL message. This process is asynchronous, meaning it's possible for cancellation messages from both A and B to cross paths.

When A Initiates Cancellation:
A sends a control message with the following structure:
Envelope: [VID_a, VID_b, Msg]
Subtype: REL_CANCEL
TSP-Control-Fields: NULL

When B Receives a Cancellation Request:

If the relationship is (VID_a, VID_b): B should reply with REL_CANCEL and then remove the relationship from its local relationship table.
If the relationship is <VID_a, VID_b>: B should remove the relationship but does not need to send a reply.
If the relationship does not exist or is not recognized: B should ignore the cancellation request.

8. Cryptographic Algorithms

Work in progress.

Throughout this specification, the following cryptographic functions are used:

- **Sign,** using public key signature
- **Encrypt,** using hybrid public key encryption
- **Digest**

The function **Concat** will be described in Section 9.

8.1 Public Key Signature

8.1.1 Ed25519

Ed25519 is a EdDSA signature algorithm with Curve-25519 and SHAR~~SA~~-512 as defined in IETF RFC 8032.

Ed25519 supports a stronger sense of unforgeability, namely SUF-CMA (Strong UnForgeability under Chosen Message Attack).

TSP implementations must support Ed25519. An example of such implementation is NaCL Ed25519 signature in open source software *libsodium*.

8.2 Public Key Encryption

TSP uses strong public key encryption schemes that supports IND-CCA2 (Indistinguishability under Adaptive Chosen Ciphertext Attack). These schemes are also called Integrated Encryption Schemes (IES), or ECIES if using Elliptic Curves, or Hybrid Public Key Encryption (HPKE) since they combine public key cryptography with the efficiency of symmetric key encryption/decryption operations. These schemes follow similar designs that incorporate a key exchange mechanism (KEM), a key derivation function (KDF), and a symmetric encryption

scheme using the ephemeral derived key, or formalized as an Authenticated Encryption with Associated Data (AEAD) function.

8.2.1 Libsodium Sealed Box

This method is recommended for initial implementations of the TSP.

Libsodium's sealed box defines these two operations.

- `crypto_box_seal(c, m, mlen, pk)`
- `crypto_box_seal_open(m, c, clen, pk, sk)`

The sender uses function `cryto_box_seal()` to encrypt a message `m` of length `mlen` for the receiver whose public key is `pk`. It outputs the resulting ciphertext of length `crypto_box_SEALBYTES + mlen` into buffer `c`.

The receiver uses function `crypto_box_seal_open()` to decrypt the ciphertext `c` of length `clen`, using the receiver's public and private key pair `(pk, sk)` and outputs the decrypted message into `m` which has length of `clen - crypto_box_SEALBYTES`.

The sealed box is a concatenation of two parts: `{ephemeral_pk, crypto_box(m, recipient_pk, ephemeral_sk, nonce}`, where ephemeral_pk and ephemeral_sk are generated by the function `crypto_box_keypair()`, and `nonce` is generated by `blake2b({ephemeral_pk, recipient_pk}).`

The `crypto_box()` function uses X25519 (RFC7748) for DH key exchange (KEM) and XSalsa20-Poly1305 / ChaCha20-Poly1305 for authenticated encryption (AEAD).

Please refer to the libsodium open source software and its associated documentation for details.

TODO: add/revise recommended settings more precisely with interoperability expectations.

8.2.2 Hybrid Public Key Encryption (HPKE) Auth Mode

HPKE is a draft standard defined in IETF RFC 9180 which formalizes and generalizes similar schemes or implementations that supports encryption of messages for a receiver with a public-private key pair and is IND-CCA2. The HPKE base mode does not use sender authentication in the HPKE itself. The algorithms in a HPKE suite are KEM (Key Exchange Mechanism), KDF (Key Derivation Function), and AEAD (Authenticated Encryption with Associated Data function).

HPKE Auth Mode configuration as defined in RFC 9180:

- KEM code: 0x0020 , DHKEM(X25519, HKDF-SHA256)
- KDF code: 0x0001 , HKDF-SHA256
- AEAD code: 0x0003, ChaCha20Poly1305

8.3 Digest Function

- Blake2b
- SHA256

TODO

9. Serialization and Encoding

Work in progress.

A TSP message is composed of the envelope sections (both the S Envelope and the ETS Envelope) and the message content. The envelope contains cryptographic elements such as VIDs, ciphertext, and signatures. The message payload can be encoded using various common formats like JSON, CBOR, MessagePack, or other schemes defined by higher-level applications.

To satisfy the need for flexibility and the performance demands of processing cryptographic elements, TSP adopts the CESR encoding scheme for conversion between text and binary formats in both directions. CESR encoding's self-framing and composability properties ensure that data object boundaries are preserved in both binary and text forms. Consequently, all messages within this specification are presented as sequential concatenations of text strings.

9.1 Base Message

9.1.1 CESR code

9.1.2 VID

9.1.3 Message Plaintext

9.1.4 Plaintext Signature

9.1.5 Ciphertext

9.1.6 Signature

9.2 Nested Message

10. Transports

Work in progress.

The TSP message envelopes are mostly agnostic to transport mechanisms which deliver them from a sender to a receiver endpoint. The authenticity, confidentiality and privacy properties of the TSP are designed to be independent of the choice of transport layer. This is one of the main goals of the TSP. That being said, it does not mean that the choice and implementation of transport mechanisms are not important to the proper function of the TSP. In this section, we define the service interface between TSP and the transport layer, and specify or clarify the most important aspects of how various transport mechanisms can be used to carry TSP message envelopes.

10.1 Transport Service Interface

10.2 Transport Mechanisms

10.2.1 HTTPS

10.2.2 WebSocket

110.2.3 TBD...

11. Acknowledgement

12. References

12.1 Normative References

1. Composable Event Streaming Representation, https://weboftrust.github.io/ietf-cesr/draft-ssmith-cesr.html
2. RFC 2119
3. RFC 8174
4. Libsodium
5. RFC 9180

12.2 Informative References

6. XYZ

Appendix

A. Comparison of notations