TLS Working Group   TLS Pathsec Protocol   Joseph Hui
INTERNET-DRAFT   Digital Island
Expires March, 2002   September, 2001
Intended Category: Standards track

TLS Pathsec Protocol

<draft-ietf-tls-pathsec-00.txt>

Status of this Memo

Abstract

The TLS Pathsec Protocol (or Pathsec Protocol in short) extends the TLS protocol into securing data in transit not only between two end points, but also between the intermediaries en route, based on TLS 1.0 with appropriate extensions that include injecting source routing policies above the Transport layer.

A typical Pathsec session comprises several sub-sessions, each of which is a TLS session with Pathsec extended semantics. It involves a client, a server, one or more intermediaries, and three individually secured channels for data and signal transports.

Integral to the Pathsec protocol are audit and opt-out features. The client or the server may selectively monitor the fidelity of the data arriving at the destination after (the data) having undergone purposed transformations performed by authorized and authenticated intermediaries designated in a routing metric; and if either end point finds the data exceedingly distorted, it may opt out gracefully.

Table of Contents

1 Introduction

This document describes the TLS Pathsec Protocol (or Pathsec Protocol in short) which extends the TLS protocol into securing data in transit not only between two end points, but also between the intermediaries en route.

Based on TLS 1.0 [TLS1] with extensions defined in [TLSX] and inspired by IP source routing [IP,STEVENS,XMPR], Pathsec in general emulates the well established end-to-end model, with augmentation for injecting routing policy necessary for traversing designated intermediaries where data may undergo authorized transformation. Thus Pathsec, with security and robustness being the paramount goals, embeds no more intelligence than what is necessary for securing the payloads entrusted by both the client and its server during a secured session. For example. In a Pathsec session, Pathsec uses routing metrics for specifying the hops between end points and the orders of traversal; but the construction of the metrics, such as by provisioning or by discovery, is outside the scope of Pathsec.

Pathsec is designed to be well suited for the request-response computing model where a client, a server, and zero or more intermediaries dot a linear processing path. Finite loops in a processing path are permissible, as they can be unfolded to form a linear pattern in Pathsec Routing Metrics.

A typical Pathsec session comprises several sub-sessions, of which each is a TLS session with Pathsec extended semantics. It involves a client, a server, one or more intermediaries, and three individually secured channels for data and signal transports. The server and all intermediaries are individually authenticated according to the TLS protocol.

Integral to the Pathsec protocol is an audit feature that allows the client or the server to selectively verify the fidelity of the data arriving at the destination. The feature is based on a "trust-but-verify" principle, for monitoring whether the extent of data distortion, which is the direct result of well-intended transformations performed by authorized and authenticated intermediaries designated in a routing metric, is within the limits of tolerance.

Also integral to the Pathsec protocol is an opt-out feature that allows the client or the server, during a session, at unilateral discretion, gracefully, to opt out of Pathsec mode and switch into the conventional TLS mode, or to opt out of the session entirely, i.e. to abort the session in progress.

Not unlike TLS or any cryptosystem, a Pathsec session is susceptible to catastrophic failure in the face of attacks aided by, for instance, compromised session key, compromised private key, compromised master secret, compromised pre-master secret, or security negligence.

As a Pathsec session involves more hops than a conventional TLS session does, it inevitably presents a larger target for attackers, even though all hops are meant to be equally securable by design; thus, it is imperative that Pathsec practitioners (in implementation and in deployment) abide by the specification in strictest adherence.

It is conceivable that Pathsec may, with reference to the end-to-end model, evolve into covering virtual end points, which may be surrogates or proxies of origin servers or user agents, in a secured content processing context.

To the TLS protocol semantics, Pathsec adds a "pathsec_signal(120)" TLS alert, a "notification" alert level, an optional "extension" element to the Alert struct (for piggy-backing supplemental data for alert processing), and a pathsec_rm(6) extension to ClientHello and to ServerHello (for facilitating Pathsec source routing).

IANA may be requested to assign a default port for Pathsec Intermediaries.

1.1. Venue of Discourse

Please send comments on this document to the IETF TLF working group's mailing list, at the writing of this document:

ietf-tls@lists.certicom.com ,

or directly to the author if the sender prefers:

jhui@digisle.net .

2 Terminology

data fidelity
The quality of data arriving at the destination of a content delivery path, measurable either quantitatively or qualitatively against the data at the source of the same content delivery path, for determining the extent of distortion.

data integrity
The quality of data arriving intact at the destination of a content delivery path. That is. if measured in terms of data fidelity, absolute data integrity means zero distortion. Message Digests are usually used for verifying data integrity.

inbound/outbound
Inbound and outbound refer to the request and response paths for messages: "inbound" means "traveling toward the origin server," and "outbound" means "traveling toward the user agent." [HTTP]

In Pathsec, "inbound" means "traveling toward the Pathsec server, which may be an origin server or its surrogate/proxy," and "outbound" means "traveling toward the Pathsec client, which may not necessarily be a user agent."

(Pathsec) Channels
There are three duplex communication channels in a Pathsec Session: 1) the Main Channel; 2) the Outbound Channel; and 3) the Inbound Channel. Ref: Figure 1.
*** Forward Compatibility Note:
*** Pathsec may in the future support multiple Outbound Channels.

(Pathsec) Client
An end point in a Pathsec session. A Pathsec client is usually a user agent, but may also be some other application entity, such as a caching proxy in a content delivery network.

(Pathsec) Hop
The direct path between two (Pathsec) nodes.

(Pathsec) Inbound Channel (IC) An inbound [HTTP] data channel from the client to the server, with one or more intermediaries en route. The hop connecting any two adjacent nodes is secured by a Pathsec Sub-session, in the form of a TLS session. Thus, an Inbound Channel is a chain of Pathsec Sub-sessions, starting at the client and ending at the server.

(Pathsec) Inbound Intermediary (II)
An intermediary in an Inbound Channel, identifiable in an Inbound

Routing Metric. The numbering of Inbound Intermediaries always starts from the client. For example, the first II immediately next to the client is II1.

(Pathsec) Inbound Routing Metric (IRM)
An Inbound Routing Metric designates the hops from the client to the server, using a strict/loose source routing policy.

(Pathsec) Main Channel (MC)
A Pathsec Sub-session, in the form of a TLS session, between the client and the server, with no intermediaries involved.

(Pathsec) Node
A Pathsec client, server, or intermediary.

(Pathsec) Outbound Channel (OC)
An outbound [HTTP] data channel from the server to the client, with one or more intermediaries en route. The hop connecting any two adjacent hops is secured by a Pathsec Sub-session, in the form of a TLS session. Thus, an Outbound Channel is a chain of Pathsec Sub-sessions, starting at the server and ending at the client.

(Pathsec) Outbound Intermediary (OI)
An intermediary in an Outbound Channel, identifiable in an Outbound Routing Metric. The numbering of Outbound Intermediaries always starts from the client. For example, the first OI immediately next to the client is OI1.

(Pathsec) Outbound Routing Metric (ORM)
An Outbound Routing Metric designates the hops from the server to the client, using a strict/loose source routing policy.

(Pathsec) Routing Metrics
There are two types of Pathsec Routing Metrics: 1) Outbound Routing Metrics; and 2) Inbound Routing Metrics.

(Pathsec) Server
An end point in a Pathsec Session. A Pathsec server is usually an origin server but may also be some other application entity, such as an origin server's surrogate (or proxy).

(Pathsec) Signal
A Pathsec Signal is issued by a client, a server, or an intermediary in the form of a TLS alert. A Pathsec signal may be accompanied by supplemental message(s), synchronously.

(Pathsec) Session and Sub-session
A Pathsec Session is comprised of one or more Pathsec Sub-

sessions, of which each is secured in the form of a TLS session between two adjacent Pathsec nodes.

(Pathsec) Sub-session Key
The TLS session key set for a Pathsec Sub-session, shared by two connecting Pathsec nodes.

relay
An intermediary that relays data or signal between client and server.

upstream/downstream
Upstream and downstream describe the flow of a message: all messages flow from upstream to downstream. [HTTP]

virtual end point
A virtual end point -- with reference to the end-to-end y -- is a surrogate (or proxy) of a server or client. It is a terminal in a processing path (that involves a client, a server, and zero or more intermediaries).

2.1 Key Word Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [KWORD].

2.2 Data Type Conventions

All data types specified in this document are to be interpreted as described in RFC-1832 [XDR] and RFC-2246 [TLS1].

3 Pathsec Session

```
    +-----------------------------------------------------------------+
    |    Client (C)                                                   |
    |           if (verify(R,R'') > limit_of_tolerence) opt_out();    |
    +-----------------------------------------------------------------+
      | ^   | ^    ^ |             |                        ^
      | |   | |    | |             |Q                      8|
      | |   | |    | |            3|                        |R''
      | |   | |    | |             v                        |
      | |   | |    | |     +----------------+       +----------------+
      | |   | |    | |     |    Inbound     |       |    Outbound    |
      | |   | |    | |     | Intermediary 1 |       | Intermediary 1 |
     1| |   | |    | |     |     (II1)      |       |     (OI1)      |
      | |   | |    | |12   +----------------+       +----------------+
      | |2  | |    | |             |                        ^
      | |   | |    | |R''          |Q'                     7|
      | |   | |    | |            4|                        |R'
      | |   | |  11| |             v                        |
      | |   | |    | |     +----------------+       +----------------+
      | |   | |   A| |     |    Inbound     |       |    Outbound    |
      | |   | |    | |     | Intermediary 2 |       | Intermediary 2 |
      | |   | |10  | |     |     (II2)      |       |     (OI2)      |
      | |  9| |    | |     +----------------+       +----------------+
      | |   | |R   | |             |                        ^
      | |  V| |    | |             |Q''                    6|
      | |   | |    | |            5|                        |R
      v |   v |    | v             v                        |
    +-----------------------------------------------------------------+
    |    Server (S)                                                   |
    |           if (audit(R,R'') > limit_of_tolerence) opt_out();     |
    +-----------------------------------------------------------------+

KEYS:
A   -- Request (from server to client) to audit responses -- R vs. R''.
Q   -- Original request/query from client.
Q'  -- Result of transforming Q by Inbound Intermediary 1 (II1).
Q'' -- Result of transforming Q' by Inbound Intermediary 2 (II1).
R   -- Original response from server.
R'  -- Result of transforming R by Outbound Intermediary 2 (OI2).
R'' -- Result of transforming R' by Outbound Intermediary 1 (OI1).
V   -- Request (from client to server) to verify responses -- R vs. R''.
Pathsec Main Channel encompasses paths: 1, 2, 9, 10, 11, and 12.
Pathsec Inbound Channel encompasses paths: 3, 4, and 5.
Pathsec Outbound Channel encompasses paths: 6, 7, and 8.
```

Figure 1: Pathsec Session Conceptual/Data Flow Diagram

```
         +--------------+
         |              |
         | Open/Re-Open |
         |              |                                     +--------+
         +--------------+                     16              |        |
                |              /----------------------------->| Close! |
               1|              |                              |        |
                v              |                              +--------+
           +----------+        |     +-------------+               ^
           |          |        |     |             |               |
           | SetUp-MC |--------/     | TearDown-OC |               |
           |          |              |             |               |
           +----------+              +-------------+               |
                |                         |   ^                    |
               2|                       13|   |12                  |
                v                         v   |                    |
           +----------+  10   +-------------------------+          |
           |          |<------|                         |          |
           | SetUp-OC |       |       In-Session        |          |6
           |          |------>|                         |-----\    |
           +----------+  11   +-------------------------+     |    |
            |      |            | ^       |   ^               |    |
            |      |            | |     14|   |15             |    |
            |      |            | |4      v   |               |    |
            |      |            | |  +-------------+          |    |
            |     3|           9| |  |             |          |5   |
            |      |            | |  | TearDown-IC |          |    |
           7|      v            | |  |             |          |    |
            | +----------+      | |  +-------------+          |    |
            | |          |<-----/ |                           v    |
            | | SetUp-IC |--------/        8            +--------------+
            | |          |------------------------------>|              |
            | +----------+                               | TearDown-All |
            \------------------------------------------->|              |
                                   7                     +--------------+
      KEYS:
      Label    Alert/Signal                Label    Alert/Signal
       1  -- pathsec_set_up_mc*, or nill  10  -- pathsec_set_up_oc*
       2  -- pathsec_set_up_oc*           11  -- pathsec_oc_set_up*
       3  -- pathsec_set_up_ic*           12  -- pathsec_tear_down_oc*
       4  -- pathsec_ic_set_up*           13  -- pathsec_oc_torn_down*
       5  -- pathsec_tear_down_all*       14  -- pathsec_tear_down_ic*
       6  -- close_notify#, or nill       15  -- pathsec_ic_torn_down*
       7  -- pathsec_tear_down_all*       16  -- close_notify#, or nill
       8  -- pathsec_tear_down_all*
       9  -- pathsec_set_up_ic*      # Existing TLS Alert  * Pathsec Signal
                                                           ! Terminal State
                         Figure 2: Pathsec State Machine
```

Frequent referals to Figures 1 and 2 are deemed helpful for the discussions throughout the document.

A Pathsec Session comprises one or more Pathsec Sub-session. A Pathsec Sub-session, secured in the same manner as a conventional TLS session with Pathsec-extended semantics, is held between two adjacent nodes along a Pathsec Channel.

Architecturally, a Pathsec Session is conducted over three secured communication channels: the Main Channel; the Outbound Channel; and the Inbound Channel. The channels are constructed during a Pathsec Set-up, which occurs at the start of the session, or in the middle of the session at the cue of Pathsec signals. The Main Channel connects the server and the client directly. The Outbound Channel carries outbound data from the server to the client through one or more intermediaries. Conversely, the Inbound Channel carries inbound data from the client to the server through one or more intermediaries. The existence of the Main Channel is mandatory. The Outbound and Inbound channels are optional. In the absense of the Outbound Channel, the Main Channel takes over its functionality in the secured delivery of outbound data, e.g. server responses. In the absense of the Inbound Channel, the Main Channel takes over its functionality in the secured delivery of inbound data, e.g. client requests. In the absense of both Outbound and Inbound Channels, the Pathsec session is operating in TLS mode, i.e. just like a conventional TLS session, with the exception that a Pathsec signal -- pathsec_set_up_oc or pathsec_set_up_ic -- may switch the session into Pathsec mode.

Pathsec signals are carried individually in a Pathsec extended TLS alert: pathsec_signal.

A Pathsec Session is always started by the client (at the Open/Re-open state in the Pathsec State Machine. [Ref:Fig 2]).

A Pathsec Session is set up by the construction of the Main, Outbound, and Inbound Channels, undergoing a series of state transitions: SetUp-MC -> SetUp-OC -> SetUp-IC -> In-Session. [Ref:Fig 2]

While Pathsec is In-Session, either the client or the server MAY signal its counterpart to tear down the OC or the IC, or to set up the OC or the IC if none exists. In addition, Audit or Verify signals MAY also be sent by the server or the client, respectively. [Ref:3.8.1,3.8.2]

The closure of a Pathsec Session, either by natural ending or by opt-out, is preceded by a TearDown-All process, which MUST sequentially close down the Inbound Channel, the Outbound Channel,

and the Main Channel. [Ref:3.9,3.10]

A closed Pathsec Session MAY be re-opened in manner similar to resuming a TLS session. [Ref:3.11]

3.1 Pathsec Main Channel

The Main Channel (MC) is defined by a Pathsec sub-session in the form of a TLS session between the client and the server.

There MUST NOT be any intermediary in MC.

As a part of Pathsec Set-up, the construction of MC starts from the client, in the same manner as starting a TLS handshake, whose successful conclusion marks the establishment of MC, which MAY be optionally milestoned by the server issuing to itself and to the client a pathsec_mc_set_up signal.

All TLS alerts, including Pathsec signals, may travel bi-directionally in MC.

In the absence of OC, outbound data, such as application server responses, travel in MC.

In the absence of IC, inbound data, such as application client requests, travel in MC.

The server's responses to a client's request to verify data fidelity travel in MC.

The client's responses to a server's request to audit (data fidelity e.g.) travel in MC.

A fatal alert affecting MC SHALL always result in the closure of the entire Pathsec Session.

MC MUST not share its pre-master and master secrets with OC.

MC SHOULD NOT share its pre-master and master secrets with IC.

3.2 Pathsec Outbound Channel

The Outbound Channel (OC) is defined by a chain of Pathsec sub-sessions in the form of hop-by-hop TLS sessions between the client and the server, inclusively.

There SHOULD be one or more intermediaries in OC. There MAY also be zero intermediary in OC, i.e. a single-hop OC where the client also plays the role of OI1 and OIn. (OI1 is the intermediary that is next to the client in OC. OIn is the intermediary that is next to the server in OC.) The number of intermediaries is limited only by the size of the hopList element in the PathsecRoutingMetric data structure. [Ref:3.4]

Note that when speaking of source routing in Pathsec, the client is always the source, where the channel construction (of hops eventually linking to the server) starts, irrespective of the channel being OC or IC. Once the channel has been set up, the concept of routing ceases to exist in a Pathsec node, which cares only to read from upstream and write to downstream.

The number and the sequence of hops, as well as other route properties, are defined in an ORM. The client and the intermediaries MAY or MAY NOT modify certain aspects of the ORM, dependent upon the ORM properties specified by the server and the client.

The ORM is carried outbound (via MC) in a TLS ServerHello pathsec_rm extension, or inbound (via OC) in a TLS ClientHello pathsec_rm extension, or in pathsec_signal_data accompanying a pathsec_set_up_oc signal (via MC). [Ref:3.4,3.6.1,TLSX]

All TLS alerts, including Pathsec signals, may travel in OC, in any direction.

Outbound application data, such as application server responses, travel in OC.

The construction of OC is hop-by-hop, with the first hop starting from the client to OI1. During the client-OI1 TLS handshake, the ORM is passed from the client to OI1. Upon completion of the first hop, OI1 connects to the next intermediary, if any, designated in ORM, and iterates the Pathsec-augmented TLS handshake with OI2, passing along the ORM. The iteration ends at the last hop with the server as the terminus.

The ShareMasterSecret property in ORM indicates if master secret is shared.

If all nodes in OC are to share a common master secret, then the client is responsible for propagating a fixed set of keying material -- pre-master secret, client random, and server random towards the server. All nodes belonging to the channel are to use such set of keying material for sub-session key generation.

If the nodes in OC do not share a common master secret, then each sub-session in OC holds its own keying secrets.

OC SHOULD NOT share its pre-master and master secrets with IC.

Prior to transporting application data in OC, the server MUST first audit the channel, by sending the client via MC a pathsec_ping signal that designates OC as the echo channel. The client MUST reply with a pathsec_echo via OC. By comparing the PathsecPing and PathsecEcho supplemental data accompanying the signals, the server is able to authenticate the channel. Upon positive authentication, the server sends the client a pathsec_echo_ok signal; otherwise, a fatal TLS alert is raised. [Ref:3.6.1]

3.3 Pathsec Inbound Channel

The Outbound Channel semantics apply equally to the Inbound Channel. (That is, 3.3 is an almost-identical twin of 3.2, substituting: inbound for outbound; IC for OC; IRM for ORM; II1, II2, IIn for OI1, OI2, OIn, respectively; "client requests" for "server responses;" and pathsec_set_up_ic for pathsec_set_up_oc.)

Whereas OC SHOULD NOT share its pre-master and master secrets with IC, IC MUST NOT share its pre-master and master secrets with OC.

3.4 Pathsec Routing Metrics

There are two types of Pathsec Routing Metrics (RMs): 1) Outbound Routing Metrics (ORM), for routing application data from the server to the client; and 2) Inbound Routing Metrics (IRM), for routing application data from the client to the server. All Pathsec Routing Metrics share an identical format and have same semantics, as in the following:

```
struct {
   RoutingPolicy        routingPolicy;
   RouteLength          routeLength;
   RoutePointer         routePointer;
   RouteDirection       routeDirection;
   ShareMasterSecret    shareMasterSecret;
   ServerMayModHopList  serverMayModHopList;
   ClientMayModHopList  clientMayModHopList;
   IntermMayModHopList  intermMayModHopList;
   opaque               serverRandom[32]; /* also serves as
                                           * channel ticket */
   opaque               pathsecReserved[64];
```

```
         opaque               hopList<1..2^14>
      } PathsecRoutingMetric;

      enum {
         loose_source_routing(0x83), /* 0x83 & 0x89 are from IP */
         strict_source_routing(0x89) /* default */
      } RoutingPolicy;

      typedef uint8 RouteLength;

      typedef unit8 RoutePointer;

      enum {
         outbound(1),
         inbound(2)
      } RouteDirection; /* no default */

      enum {
         dont_share_master_secret(0),
         share_master_secret(1) /* default */
      } ShareMasterSecret;

      enum {
         server_may_not_modify_hop_list(0),
         server_may_modify_hop_list(1) /* default */
      } ServerMayModRM;

      enum {
          client_may_not_modify_hop_list(0),
          client_may_modify_hop_list(1) /* default */
      } ClientMayModRM;

      enum {
         interm_may_not_modify_hop_list(0), /* default */
         interm_may_modify_hop_list(1)
      } IntermMayModRM;

      hopList    := hops
      hops       := hostport [ , hops ]
      hostport   := host [ : port ]
      host       := hostname | hostnumber
      hostname   := ialpha [ . hostname ]
      hostnumber := digits . digits . digits . digits
      port       := digits

      (Refer to [URI] for ialpha and digits.)

      Pathsec server and intermediaries share with TLS the same default
```

port: 443.
*** Forward Compatibility Note:
*** IANA may be requested to assign a new default port for
*** Pathsec Intermediaries.

An examples of (comma-delimited) hopList:

"I1.x.com,I2.y.com:4567,server.z.com:7890" is a three-hop list such that in an ORM, application data flow in the way of:

```
server.z.com:7890 -> I2.y.com:4567 -> I1.x.com -> client
```

and in an IRM, application data flow in the way of:

```
client -> I1.x.com -> I2.y.com:4567 -> server.z.com:7890
```

A Routing Metric (RM) may be carried in a pathsec_rm extension in a ServerHello or a ClientHello. It may also be carried in the pathsec_signal_data accompanying a pathsec_set_up_oc or pathsec_set_up_ic signal, which is delivered in a TLS alert typed pathsec_signal.

Either the server or the client MAY be the RM originator, whose wishes (as specified in routingPolicy, routeDirection, shareMasterSecret, serverMayModHopList, clientMayModHopList, intermMayModHopList, and hopList) must be respected by all nodes in a channel, with the following exceptions. The server MAY negotiate loose_source_routing to strict_source_routing; the server MAY negotiate server_may_not_modify_hop_list to server_may_modify_hop_list; the server MAY negotiate the server MAY negotiate interm_may_modify_hop_list to interm_may_not_modify_hop_list; or the server MAY negotiate share_master_secret to dont_share_master_secret. The client MUST NOT perterb the "server-side" hops specified by the server, though it MAY prepend "client-side" hops to hopList. The server SHOULD NOT perterb the "client-side" hops specified by the client, though it MAY append "server-side" hops to them.

RoutingPolicy is for the originator of an RM, usually the server but sometimes the client, to specify the routing policy governing a Pathsec channel: loose source routing, or strict source routing (the default). RoutingPolicy, once set, SHOULD NOT be changed. All nodes MUST execute the routing policy in the exact manner as described in [3.4.1] and [3.4.2]. (Also refer to [IP,STEVENS] for the workings of IP source routing.)

RouteLength specifies the number of hops in hopList, e.g. 3 for three hops.

RoutePointer points at the destination node of the current hop. RoutePointer increments by 1 per hop.

RouteDirection indicates if the route is for inbound or outbound application data. For example, if routeDirection = outbound, then the RM is for OC, i.e. ORM.

ShareMasterSecret indicates if all nodes belonging to the channel that employs the RM are to share a common master secret for keying purpose.

ServerMayModHopList indicates if the server may modify hopList.

ClientMayModHopList indicates if the client may modify hopList.

IntermMayModHopList indicates if the intermediaries may modify hopList.

ServerRandom contains the server random for keying purpose, in case of share_master_secret.

ServerRandom also serves as the channel ticket, by which the server, which may face multiple connection requests, determines which channel a connecting party belongs to during a TLS handshake.

PathsecReserved is a dummy at the writing of this document.

HopList contains the list nodes en route, in format defined above. The first node is always II1 or OI1, and the last node is always the server, because the construction of IC or OC always starts from the client. All nodes in a channel must observe the rules set in: serverMayModHopList, clientMayModHopList, and intermMayModHopList, with few forementioned server exceptions.

3.4.1 Pathsec Strict Source (and Record) Routing

In Pathsec strict source (and record) routing (PSSRR), the client of a Pathsec channel to be constructed is first given an RM, i.e. PathsecRoutingMetric, either through a ServerHello pathsec_rm extension or in the pathsec_signal_data accompanying a pathsec_set_up_ic/pathsec_set_up_oc signal, where routingPolicy is set to strict_source_routing. The RM is to be forwarded inbound through a ClientHello pathsec_rm extension during the TLS handshakes that will establish the sub-sessions in the channel.

Before forwarding the RM in a ClientHello, a Pathsec node MUST increment routePointer by 1.

Each node (in the channel) uses routePointer as the locator for determining its TLS server (in a Pathsec sub-session) to connect to, only if routePointer is not greater than routeLength. For example, if routePointer is 2, then the second node in hopList is the TLS server of the sub-session to be established. If routePointer is greater the routeLength, then the end of the route has been reached. (Note that routePointer pointing at the Pathsec server does not necessarily indicate the end of the route.)

An intermediary MUST be aware that if routeLength equals routePointer in the RM given, then it is the last intermediary in the channel, e.g. OIn, or IIn, and may be called upon to perform a special task (such as flipping an authentication string) in channel authentication later. [Ref:3.6.1-pathsec_echo]

Figure 3 illustrates the algorithm of Pathsec strict source (and record) routing by example.

The recording of the route is done by leaving hopList alone and incrementing routePointer properly in each hop.

3.4.2 Pathsec Loose Source (and Record) Routing

Pathsec loose source (and record) routing (PLSRR) works in similar ways as PSSRR does, with the crucial exception that the client or an intermediary may, if permitted by the client_may_modify_hop_list or interm_may_modify_hop_list respectively, properties in the RM, insert hops between the Pathsec server and itself. (The routingPolicy in the RM MUST be pre-set to loose_source_routing prior to channel construction.)

Figure 4 illustrates the algorithm of Pathsec loose source (and record) routing by example.

The recording of the route is done by properly updating hopList and routeCount, and incrementing routePointer in each hop.

```
            +-------+
            |   C   |
            +-------+   routeLength  = 3
                |       routePointer = 1              *
                |       hopList                    = {I1,I2,S}
                |       Pathsec sub-session TLS client = C
                |       Pathsec sub-session TLS server = I1
                v
            +-------+
            |   I1  |
            +-------+   routeLength  = 3
                |       routePointer = 2                 *
                |       hopList                    = {I1,I2,S}
                |       Pathsec sub-session TLS client = I1
                |       Pathsec sub-session TLS server = I2
                v
            +-------+
            |   I2  |
            +-------+   routeLength  = 3
                |       routePointer = 3                    *
                |       hopList                    = {I1,I2,S}
                |       Pathsec sub-session TLS client = I2
                |       Pathsec sub-session TLS server = S
                v
            +-------+
            |   S   |
            +-------+   routeLength  = 3
                        routePointer = 4
                        hopList      = {I1,I2,S}
```

The hopList given to Pathsec client C is {I1,I2,S} where S is the Pathsec server; I1 and I2 are intermediaries; routeLength is 3; and routePointer is initially 1.

Figure 3: Pathsec Strict Source (and Record) Routing

```
            +-------+
            |   C   |
            +-------+   routeLength  = 3
                |       routePointer = 1              *
                |       hopList                    = {I1,I2,S}
                |       Pathsec sub-session TLS client = C
                v       Pathsec sub-session TLS server = I1
            +-------+
            |   I1  |
            +-------+   routeLength  = 3
                |       routePointer = 2
                |       hopList                    = {I1,I2,S}
                |          I1 inserts I1a and I1b into hopList, then
                |       routeLength  = 5
                |       routePointer = 2                 *
                |       hopList                    = {I1,I1a,I1b,I2,S}
                |       Pathsec sub-session TLS client = I1
                v       Pathsec sub-session TLS server = I1a
            +-------+
            |  I1a  |   routeLength  = 5
            +-------+   routePointer = 3                     *
                |       hopList                    = {I1,I1a,I1b,I2,S}
                |       Pathsec sub-session TLS client = I1a
                v       Pathsec sub-session TLS server = I1b
            +-------+
            |  I1b  |   routeLength  = 5
            +-------+   routePointer = 4                         *
                |       hopList                    = {I1,I1a,I1b,I2,S}
                |       Pathsec sub-session TLS client = I1b
                v       Pathsec sub-session TLS server = I2
            +-------+
            |   I2  |   routeLength  = 5
            +-------+   routePointer = 5                            *
                |       hopList                    = {I1,I1a,I1b,I2,S}
                |       Pathsec sub-session TLS client = I2
                v       Pathsec sub-session TLS server = S
            +-------+
            |   S   |   routeLength  = 5
            +-------+   routePointer = 6
                        hopList      = {I1,I1a,I1b,I2,S}
```

The hopList given to Pathsec client C is {I1,I2,S} where
S is the Pathsec server; I1 and I2 are intermediaries;
I1a and I1b are intermediaries inserted by I1;
routeLength is initially 3, and is changed to 5 by I1;
and routePointer is initially 1.

Figure 4: Pathsec Loose Source (and Record) Routing

3.5 Pathsec Extended TLS ClientHello/ServerHello

Pathsec adds "pathsec_rm" to the existing TLS extensions defined in [TLSX]. It is to be included in ServerHello or ClientHello as applicable. [Ref:3.4]

The following list enumerates the TLS extension types defined in [TLSX] at the writing of this document, plus pathsec_rm:

```
   enum {
      dns_name(0),
      max_record_size(1),
      client_certificate_url(2),
      trusted_ca_keys(3),
      truncated_hmac(4),
      status_request(5),
      pathsec_rm(6), /* new for Pathsec */
      (65535)
   } ExtensionType;
```

Origin servers, surrogates, proxies, and user agents that do not understand the pathsec_rm extension SHOULD simply ignore the extension.

3.6 Pathsec Extended TLS Alert

The Pathsec Protocol extends the TLS Alerts data structure (defined in [TLS1]) to include an optional element: "extension." Pathsec also introduces a new alert level: "notification;" and a new alert type: "pathsec_signal." The notification level is for accommodating alerts that are difficult to precisely characterize as warning or fatal. The recipient MUST NOT ignore the alert, unless it does not support the alert type specified in the description field. The optional Alerts.extension is for piggy-backing supplemental data for alert processing. The sender and recipient(s) MUST cast the opaque Alerts.extension data into alert-type-specific data structure(s) for further processing. In the case of Pathsec, the extension data is cast into the PathsecAlert data structure defined in 3.6.1.

*** Author's Note:
*** [TLS1] did not script a forward compatibility note for alert
*** extensions; so backward compatibility issues related to an
*** extended TLS Alert struct are open at the writing of this
*** document.

Origin servers, surrogates, proxies, and user agents that do not support pathsec_signal SHOULD raise an unexpected_message alert to

the pathsec_signal sender.

The following describes the Pathsec extended TLS Alert data structure and enumerates TLS alerts, including pathsec_signal, which is an addition to the TLS alerts having been compiled in [TLSX], at the writing of this document:

```
struct {
    AlertLevel level;
    AlertDescription description;
    opaque extension<0..2^16-1>;  /* new for Pathsec */
} Alert;

enum {
   warning(1),
   fatal(2),
   notification(3), /* new for Pathsec */
   (255)
} AlertLevel;

enum {
   close_notify(0),
   unexpected_message(10),
   bad_record_mac(20),
   decryption_failed(21),
   record_overflow(22),
   decompression_failure(30),
   handshake_failure(40),
   certificate_unobtainable(41),       /* new for TLSX */
   bad_certificate(42),
   unsupported_certificate(43),
   certificate_revoked(44),
   certificate_expired(45),
   certificate_unknown(46),
   illegal_parameter(47),
   unknown_ca(48),
   access_denied(49),
   decode_error(50),
   decrypt_error(51),
   export_restriction(60),
   protocol_version(70),
   insufficient_security(71),
   internal_error(80),
   user_canceled(90),
   no_renegotiation(100),
   unsupported_extension(110),         /* new for TLSX */
   bad_extension_order(111),           /* new for TLSX */
   unrecognised_domain(112),           /* new for TLSX */
```

```
         bad_ocsp_response(113),             /* new for TLSX */
         pathsec_signal(120),                /* new for Pathsec */
         (255)
      } AlertDescription;

      [Ref:TLS1,TLSX]
```

3.6.1 Pathsec Signals

This section describes the Pathsec Signals. All Pathsec nodes MUST relay Pathsec signals downstream. A Pathsec signal affects all nodes in its path, including the end point(s) where it expires. A Pathsec signal is delivered in a pathsec_signal TLS alert, with a TLS alert extension that is to be cast into the PathsecAlert data structure defined as follows.

```
      struct {
         PathsecSignal pathsec_signal_type;
         opaque        pathsec_signal_data<0..2^15-1>;
      } PathsecAlert;

      enum {
         pathsec_set_up_mc(1),
         pathsec_mc_set_up(2),
         pathsec_set_up_oc(3),
         pathsec_oc_set_up(4),
         pathsec_set_up_ic(5),
         pathsec_ic_set_up(6),
         pathsec_tear_down_mc(7),
         pathsec_mc_torn_down(8),
         pathsec_tear_down_oc(9),
         pathsec_oc_torn_down(10),
         pathsec_tear_down_ic(11),
         pathsec_ic_torn_down(12),
         pathsec_tear_down_all(13),
         pathsec_verify_request_start(14),
         pathsec_verify_request_end(15),
         pathsec_verify_response_start(16),
         pathsec_verify_response_end(17),
         pathsec_opt_out_oc(18),
         pathsec_opt_out_oc_ack(19),
         pathsec_opt_out_oc_nack(20),
         pathsec_opt_out_ic(21),
         pathsec_opt_out_ic_ack(22),
         pathsec_opt_out_ic_nack(23),
         pathsec_source_route_failed(24),
         pathsec_feature_unsupported(25),
```

```
         pathsec_ping(26),
         pathsec_echo(27),
         pathsec_echo_ok(28),
         (255)
      } PathsecSignal;
```

pathsec_set_up_mc(1)

The client or server may optionally send pathsec_set_up_mc to itself (for invoking a pathsec_set_up_mc callback, e.g.). Upon receipt of this signal, the client or server enters the SetUp-MC state (in the Pathsec State Machine).

pathsec_mc_set_up(2)

Either the client or server may optionally send or receive pathsec_mc_set_up upon exit of SetUp-MC, which is to be followed by SetUp-OC.

pathsec_set_up_oc(3)

The server sends this signal to the client via MC, and optionally to itself. The receiver of this signal MUST enter SetUp-OC. The pathsec_signal_data accompanying this signal contains a PathsecRoutingMetric, where routeDirection = outbound, i.e. an ORM. The client MUST start constructing the OC according to the ORM. The server MUST listen for an outstanding OC connection request, at the server port specified/implied in the ORM.

pathsec_oc_set_up(4)

The server sends pathsec_oc_set_up to the client via MC, and optionally to itself, upon the completion of SetUp-OC.

pathsec_set_up_ic(5)

The server sends this signal to the client via MC, and optionally to itself. The receiver of this signal MUST enter SetUp-IC. The pathsec_signal_data accompanying this signal contains a PathsecRoutingMetric, where routeDirection = inbound, i.e. an IRM. The client MUST start constructing the IC according to the IRM. The server MUST listen for an outstanding IC connection request, at the server port specified/implied in the IRM.

The client MAY also send this signal to the server, enclosing in pathsec_signal_data an IRM with "client-side" hops. In such case, the server MAY optionally prepend "server-side" hops to the "client-side" hops, by inserting "server-side" nodes in front of

the last node in hopList. For example, "cs1,cs2,svr" becomes "cs1,cs2,ss1,ss2,svr." The server then sends back to the client a pathsec_set_up_ic, with a newly negotiated IRM if applicable.

pathsec_ic_set_up(6)

The server sends pathsec_ic_set_up to the client via MC, and optionally to itself, upon the completion of SetUp-IC.

pathsec_tear_down_mc(7)

This signal SHOULD NOT be used, pending future specification. (If MC goes, so go all channels and the entire Pathsec session. Thus pathsec_tear_down_all seems to be more appropriate for virtually all foreseeable cases at the writing of this document.)

pathsec_mc_torn_down(8)

This signal SHOULD NOT be used, pending future specification.

pathsec_tear_down_oc(9)

The server MAY at any time send via OC the client pathsec_tear_down_oc, and vice versa. Both the signal sender and receiver must enter TearDown-OC immediately. Each intermediary en route MUST immediately forward the signal downstream, and then enter TearDown-OC itself. The server and client MUST notify their respective applications of this signal, and data pending for read/write MAY be flushed.

pathsec_oc_torn_down(10)

The server sends pathsec_oc_torn_down to the client via MC, and optionally to itself, upon the completion of TearDown-OC.

pathsec_tear_down_ic(11)

The server MAY at any time send via IC the client pathsec_tear_down_ic, and vice versa. Both the signal sender and receiver must enter TearDown-IC immediately. Each intermediary en route MUST immediately forward the signal downstream, and then enter TearDown-IC itself. The server and client MUST notify their respective applications of this signal, and data pending for read/write MAY be flushed.

pathsec_ic_torn_down(12)

The server sends pathsec_ic_torn_down to the client via MC, and

optionally to itself, upon the completion of TearDown-IC.

pathsec_tear_down_all(13)

The server MAY at any time send via MC the client pathsec_tear_down_all, and vice versa. Both the signal sender and receiver must enter TearDown-All immediately. Pathsec_tear_down_all signals the imminent closure of the Pathsec session. All channels are to be torn down as soon as possible, with provision for I/O flushing as appropriate. The server and client MUST notify their respective applications of this signal, and data pending for read/write MAY be flushed.

pathsec_verify_request_start(14)

The server MAY send via MC the client, and vice versa, a pathsec_verify_request_start to initiate a process to verify the data fidelity in OC.

pathsec_verify_request_end(15)

The server MAY send via MC the client, and vice versa, a pathsec_verify_request_end to terminate the process of verifying the data fidelity in OC.

pathsec_verify_response_start(16)

The receiver of pathsec_verify_request_start responses with a pathsec_verify_response_start to signal that verification data is forthcoming.

pathsec_verify_response_end(17)

The receiver of pathsec_verify_request_end responses with a pathsec_verify_response_end to signal the end of verification data.

pathsec_opt_out_oc(18)

The server MAY send via MC the client, and vice versa, a pathsec_opt_out_oc to tear down OC. The signal sender SHOULD time out (with a pathsec_opt_out_oc_nack) if it does not receive a pathsec_opt_out_oc_ack in 10 seconds.

pathsec_opt_out_oc_ack(19)

The client or the server MUST send via MC pathsec_opt_out_oc_ack to acknowledge the receipt of pathsec_opt_out_oc, prior to

entering TearDown-OC. Upon receiving pathsec_opt_out_oc_ack, the pathsec_opt_out_oc sender SHOULD enter TearDown-OC.

pathsec_opt_out_oc_nack(20)

Upon timing out of a pathsec_opt_out_oc, the pathsec_opt_out_oc sends itself and optionally the pathsec_opt_out_oc receiver a pathsec_opt_out_oc_nack, via MC.

pathsec_opt_out_ic(21)

The server MAY send via MC the client, and vice versa, a pathsec_opt_out_ic to tear down IC. The signal sender SHOULD time out (with a pathsec_opt_out_ic_nack) if it does not receive a pathsec_opt_out_ic_ack in 10 seconds.

pathsec_opt_out_ic_ack(22)

The client or the server MUST send via MC pathsec_opt_out_ic_ack to acknowledge the receipt of pathsec_opt_out_oc, prior to entering TearDown-OC. Upon receiving pathsec_opt_out_ic_ack, the pathsec_opt_out_oc sender SHOULD enter TearDown-IC.

pathsec_opt_out_ic_nack(23)

Upon timing out of a pathsec_opt_out_ic, the pathsec_opt_out_ic sends itself and optionally the pathsec_opt_out_ic receiver a pathsec_opt_out_ic_nack, via MC.

pathsec_source_route_failed(24)

This signal SHOULD NOT be used, pending future specification. A Pathsec intermediary, in case of locally fatal error, sends a pathsec_source_route_failed in both upstream and downstream directions. This signal is fatal to the channel. Upon receiving pathsec_source_route_faile, the server and the client SHOULD independently signal pathsec_tear_down_oc (or pathsec_tear_down_ic as applicable). The client and server applications MUST be notified of the source route failure. The channel torn down MAY be re-constructed, provide at least one application layered above Pathsec commands the server or the client to signal pathsec_set_up_oc (or pathsec_set_up_ic as applicable).

pathsec_feature_unsupported(25)

A Pathsec node is being requested (by the client or the server) to perform a task it does not support, then it sends a pathsec_feature_unsupported upstream, which will be relayed to the

requester.

pathsec_ping(26) & pathsec_echo(27)

The server MAY send a pathsec_ping to the client, and vice versa, only via MC, for the purposes of: 1) the pinger inquiring the highest Pathsec version supported by the echo-er; and 2) the server authenticating a channel that might have been constructed without Client Authentication during TLS Handshake(s) earlier. (For example, a bogus last intermediary could gain "acquaintance" with a Pathsec server using replay attack with an intercepted channel ticket embedded in a ClientHello's plain-text serverRandom, if the server did not demand Client Authentication Handshake. Note that in Pathsec, the server by default does not demand Client Authentication Handshake, because the last intermediary may also be the Pathsec client (in a one-hop channel) which may happen to be a user agent, and it is not common practice that user agents are in possesion of certificates.)

The pinger packs pathsec_signal_data with a PathsecPing (defined below). The echo-er copies (or cast) a PathsecPing into a PathsecEcho (also defined below), assigning proper values to echo_major and echo_minor, and then emits the PathsecEcho (in pathsec_signal_data) via the channel indicated by echo_channel_id.

The PathsecPing sender (aka pinger) SHOULD time out, if the expected PathsecEcho fails to arrive within a reasonable time limit: 10 seconds * approximated-number-of-hops-in-echo-channel. All intermediaries relaying a PathsecEcho towards its destination, except the last intermediary next to the pinger in the echo channel, MUST NOT modify the content of a PathsecEcho.

PathsecPing and PathsecEcho are defined as follows.

```
   struct {
      uint16  ping_id;
      uint16  echo_channel_id;
      uint8   ping_major;
      uint8   ping_minor;
      unit8   echo_major;
      uint8   echo_minor;
      opaque  random[24];
  } PathsecPing;

  struct {
      uint16  ping_id;
      uint16  echo_channel_id;
      uint8   ping_major;
```

```
            uint8   ping_minor;
            unit8   echo_major;
            uint8   echo_minor;
            opaque  random[20];
        } PathsecEcho;
```

Ping_id is for tracking pings and echos. Its value is set by the pinger and MUST NOT be modified by the echo-er or relays. The value set is unique within an echo channel, and may wrap around.

Echo_channel_id indicates the channel via which the PathsecEcho MUST travel. Its value is set by the pinger and MUST NOT be modified by the echo-er or relays. There are three permanently pre-defined values: 0 -- via MC; 1 -- via OC; 2 -- via IC.
*** Forward Compatibility Note:
*** Future Pathsec versions may support more than three channels.

Ping_major and ping_minor indicate the highest major and minor numbers of the Pathsec version the pinger supports, starting from major 1, minor 0. The pinger MUST instantiate ping_major and ping_minor with correct values; and set echo_major and echo_minor to 0.

Echo_major and echo_minor indicate the highest major and minor numbers of the Pathsec version the echo-er (of a PathsecPing) supports, starting from major 1, minor 0. The echo-er, who is the originater of a PathsecEcho in reply to a PathsecPing, MUST instantiate echo_major and echo_minor with correct values.

Major number being 0 indicates the version is experimental. Experimental versions MUST have non-zero minor numbers.

PathsecEcho.random contains 20 random bytes copied from EchosecPing.random, which was generated by the pinger, for the purpose of authenticating the echo channel. The last intermediary in the echo channel MUST reverse the byte sequence of PathsecEcho.random, i.e. the first byte becomes the last, the last byte becomes the first, and so on, prior to forwarding the PathsecEcho to its destination -- the server.

pathsec_echo_ok(28)

The pinger MUST keep a copy of the PathsecPing sent. Upon receipt of a PathsecEcho, the pinger MUST compare the ping_id and echo_channel_id in the PathsecPing and PathsecEcho for identical matches. Additionally, if the echo channel is not MC (i.e. echo_channel_id != 0), then the pinger MUST reverse the byte sequence in PathsecEcho.random and compare PathsecPing.random to

PathsecEcho.random. If they are equal, then the echo channel is authenticated, and a pathsec_echo_ok signal is to be sent over MC to the echo-er, accompanied by the PathsecEcho with the PathsecEcho.random in original byte sequence (originally set by the pinger). Otherwise, the last intermediary is deemed an imposter, because it has failed to decipher the PathsecEcho (in order to reverse the bytes in PathsecEcho.random); and an "insufficient_security" TLS fatal alert MUST be raised. Application data SHOULD NOT travel in OC or IC unless the channel in question has been "certified" for use by a pathsec_echo_ok.

The pinger MAY discard the PathsecPing copy it keeps after processing the corresponding PathsecEcho.

3.7 Pathsec Set-up

The Pathsec Set-up involves three major steps of state transitions:

```
Open/Re-open ->
   SetUp-MC -> SetUp-OC -> SetUp-IC ->
      In-Session
```

[Ref:Fig 2]

Step 1: the client, in SetUp-MC state, initiates connection to the server to establish the the Main Channel, using TLS handshake with Pathsec-extended ClientHello and ServerHello. [Ref:3.1,3.5;3.6.] In case of a fatal alert, the session -- server and client -- transits to the Close state; else, the session enters SetUp-OC.

Step 2: the client, in SetUp-OC state, scans the ServerHello extensions for ORM. If one exists, then it proceeds to set up the Outbound Channel. Using the ORM embedded in a ServerHello extension as the guideline, it initiates connection to the first Outbound Intermediary (the OI1 designated in the ORM), which in turn initiates connection to the next OI (if one exists), and so on, eventually connecting to the server. [Ref:3.2] In case of a fatal error, the session -- client, server, and all intermediaries -- enters TearDown-All state; else, the session enters SetUp-IC state.

Step 3: the client, in SetUp-IC state, scans the ClientHello extensions for IRM. If one exists, optionally sets up the Inbound Channel. Using the Inbound Routing Metric embedded in a ClientHello Extension, which the client has previously sent to the server while setting up the Main Channel, it (the client) initiates connection to the first Inbound Intermediary (i.e. the II1 designated in the IIM), which in turn initiates connection to the next II, and so on,

eventually connecting to the server. [Ref:3.3]

The successful completion of a Pathsec Set-up is always followed by In-Session state. [Ref:Fig 2]

3.8 Pathsec In-Session

When a Pathsec session is in In-Session state, application data flow is guaranteed.

Alerts and signals flow freely at any time.

During In-Session, the server MAY at any time via MC send the client a pathsec_set_up_oc or a pathsec_set_up_ic signal to cause both the server and the client to enter SetUp-OC or SetUp-IC, respectively.

During In-Session, the server MAY at any time via MC send the client, or vice versa, a pathsec_tear_down_oc or a pathsec_tear_down_ic signal to cause both the server and the client to enter TearDown-OC or TearDown-IC, respectively.

During In-Session, the server MAY at any time via MC send the client, or vice versa, a pathsec_tear_down_all signal to cause both the server and the client to enter TearDown-ALL.

The following signals always bring the Pathsec session back to In-Session: pathsec_oc_set_up, pathsec_ic_set_up, pathsec_oc_torn_down, and pathsec_ic_torn_down,

During In-Session, the arrival of verify/audit and opt-out signals SHALL cause no state transition.

[Ref:3.6.1]

A multi-threaded implementation MAY, in the interest of optimizing application data throughput, off-load signal handling, which often requires the session to enter a new state (e.g. SetUp-IC) and then to return to In-Session. However, the implenmentor is responsible for synchronizing the In-Session thread with the off-loaded signal thread(s) such that there MUST NOT be dead-locking or inconsistency in payload presentatiion (to the application layered above Pathsec).

3.8.1 Pathsec Verify

A Pathsec Verify is always initiated by the client. (If initiated by the server, then it is termed Pathsec Audit.)

The client MAY at any time send a pathsec_verify_request_start signal to the server via MC, in order to verify the data fidelity in OC, per request from its appication above the Pathsec layer. The server MUST signal its own application layered above Pathsec to start a data verification process. The verification data, which is identical to the data that the server releases into OC, is transmitted over MC.

The server, commanded by its application layered above, signals the client that verification data is forthcoming with a pathsec_verify_response_start.

The server and client applications MUST device their own means for delimiting the data being verified, e.g. starting from the next HTTP response.

The data verification request is in force until the client signals the server with a pathsec_verify_request_end.

The data verification response is in force until the server signals the client with a pathsec_verify_response_end.

3.8.2 Pathsec Audit

A Pathsec Audit is always initiated by the server. (If initiated by the client, then it is termed Pathsec Verify.)

A Pathsec Audit may take one of two forms: 1) verifying the data fidelity in OC; or 2) authenticating IC or OC.

The server MAY at any time send a pathsec_verify_request_start signal to the client via MC, in order to verify the data fidelity in OC, per request from its appication above the Pathsec layer. The client MUST signal its own application layered above Pathsec to start a data verification process. The verification data, which is the data that the client receives from OC, is transmitted over MC.

The client, commanded by its application layered above, signals the server that verification data is forthcoming with a pathsec_verify_response_start.

The server and client applications MUST device their own means for delimiting the data being verified, e.g. starting from the next HTTP response.

The data verification request is in force until the server signals the client with a pathsec_verify_request_end.

The data verification response is in force until the client signals the server with a pathsec_verify_response_end.

[Ref:3.6.1]

Refer to the pathsec_ping, pathsec_echo, and pathsec_echo_ok sub-sections in [3.6.1] for the details of authenticating an inbound/outbound channel without using certicate or password.

3.8.3 Pathsec Opt-out

Either the client or the server MAY opt out of OC, or IC, or the entire Pathsec session, at any time, without cause, by raising pathsec_opt_out_oc, pathsec_opt_out_ic, or pathsec_tear_down_all, respectively.

Refer to the raising pathsec_opt_out_oc, pathsec_opt_out_oc_ack, pathsec_opt_out_oc_nack, pathsec_opt_out_ic, pathsec_opt_out_ic_ack, pathsec_opt_out_ic_nack, pathsec_tear_down_all, respectively. and pathsec_tear_down_all sub-sections in [3.6.1] for the workings of opt-out signal processing.

The opt-out feature is not available for intermediaries.

A Pathsec implementation MUST provide the necessary API for applications layered above Pathsec to exercise opt-outs.

3.9 Pathsec Tear-down

All nodes in an IC or OC receiving a pathsec_tear_down_ic or pathsec_tear_down_oc respectively MUST propagate the received signal downstream, and then proceed to close down its upstream and downstream connections. The server and the client should signal themselves with pathsec_ic_torn_down or pathsec_oc_torn_down as appropriate.

3.10 Pathsec Close

The closure of a Pathsec session SHOULD be preceeded by the tear-downs of the channels, in strict sequence: IC, OC, and MC.

3.11 Pathsec Re-open

A naturally closed Pathsec session, i.e. the closure was not due to a

fatal alert, MAY be re-opened in manner similar to resuming a TLS session, using section ID as resumption hint. In order to support Re-open, both the client and the server MUST be able to cache the routing metrics of a resumable session off-line.

4 Pathsec Extensions to TLS

Refer to sections 3.5 and 3.6.

5 Application Considerations

Pathsec, co-locating with TLS (above the transport layer) of the OSI stack, is semantically indifferent to the payload it carries.

Nonetheless, Pathsec is designed to be well suited for the request-response computing model where a client, a server, and zero or more intermediaries dot a linear processing path. Finite loops in a processing path are permissible, as they can be unfolded to form a linear pattern in a Pathsec Routing Metric.

It is conceivable that Pathsec MAY be used by applications that involve value-added services provided by intermediaries trusted and verified by servers and clients. It MAY also be used by content delivery networks (CDNs) for transporting secured payloads, such as propagating secured resource updates, say, multicasting authenticated cache invalidation signals from an origin server to its caching proxies. (For reference of application models that are being discussed by IETF working groups and may find Pathsec relevant, consult literature in [OPES], [WEBI], [CDI].)

It is conceivable that Pathsec may evolve into covering virtual end points -- in end-to-end simile -- which may be surrogates and proxies of origin servers or user agents, in a secured content processing context.

6 Security Considerations

Unless stated otherwise, all failure modes discussed in this section are catastrophic, though variable in scope of damage. They all warrant fatal alerts, in spite some damaged sessions may be salvageable. The server MAY opt to NOT salvage a salvageable session without cause. Note that detection of failure modes discussed herein is outside the scope of the Pathsec protocol.

All Pathsec practitioners (in implementation and in deployment)

SHOULD be well acquainted with historic and up-to-date issues related to network, data, and system security. [Ref:DENNING,NICHOLS,RESCORLA,STARTLS,TLS1,TLSX]

6.1 Compromised Private Key

If the private key of a Pathsec node is compromised, then the Pathsec Channel involving the compromised node is also compromised for good and MUST be torn down.

If the compromised node is either the client or the server, then the session is compromised. All nodes in session MUST enter the TearDown-All state, to be followed by Close. The routing metric containing the compromised node is compromised indefinitely, until a new and valid private key is available. The server (and the client if applicable) MUST mark the routing metric unusable until proper key replenishment.

If the compromised node is an intermediary, then the session may be salvageable, only by the server. If the compromised metric can be replaced by an alternative one or be repaired with a new private key, then the server MUST issue pathsec_tear_down_oc or pathsec_tear_down_ic as appropriate, and all nodes in the damaged channel MUST enter TearDown-OC (or TearDown-IC as appropriate), and then return to In-Session. After receiving a pathsec_oc_torn_down (or pathsec_ic_torn_down) from the client, the server MAY signal pathsec_set_up_oc (or pathsec_set_up_ic) to lead the session into SetUp-OC (or SetUp-IC), and In-Session next.

Salvaging a private-key-compromised Pathsec Session without sufficient justification (which is outside the Pathsec scope) is NOT RECOMMENDED.

6.2 Compromised Sub-session Key

If a sub-session key is compromised, then an attacker may conduct man-in-the-middle activities in the channel involving the compromised hop. The scopes of damage due to compromised sub-session key range from per sub-session to per session. However, keep in mind that compromised sub-session key may only be symptomatic to compromised private key(s), compromised master secret, or compromised pre-master secret.

6.3 Compromised Master Secret

If the master secret, originated from the client during client-server handshake, is compromised, then an attacker may derive Sub-session Key(s) shared by any two adjacent Pathsec nodes using the publicly available key derivation function (KDF). (The KDF takes the captured master secret and two random blocks separately generated by the client and the server during handshake as input parameters.) Because the randoms are always transmitted in plain text in TLS, they are fairgames to network snoopers. The scope of damage due to compromised master secret is per session.

6.4 Compromised Pre-Master-Secret

If the pre-master secret, originated from the client during client-server handshake, is compromised, then an attacker may derive the master secret (and thus sub-session key(s)) using the publicly available key derivation function (KDF). (The KDF takes the captured pre-master secret and two random blocks separately generated by the client and the server during handshake as input parameters.) Because the randoms are always transmitted in plain text in TLS, they are fairgames to network snoopers. The scope of damage due to compromised pre-master secret is per session.

6.5 Ciphersuite Degradation

Each intermediary of an Outbound Channel or an Inbound Channel SHOULD support at least one ciphersuite that is functionally equivalent to and is at least as strong as the one deployed in the Main Channel. Otherwise, the session is vulernable to downgrade attack.

6.6 Perils of Sharing Master Secret Across Channels

The sharing of a master secret (or pre-master secret in a similar vein) across-channel SHOULD NOT be allowed. For instance, the master secret of the Main Channel or the Inbound Channel MUST NOT be shared with the Outbound Channel. Otherwise, outbound intermediaries, say language translaters or ad inserters, may derive the necessary sub-session key(s) to snoop inbound traffic, which may contain passwords that outbound intermediaries are not privy to.

All nodes of a Pathsec session MUST know that both the server and the client know the common master secrets of all channels.

6.7 Intermediary Weakness

The data in transit to and from the call-outs and value-added services fashioned by a Pathsec intermediary MUST be secured with a cryptosystem that is at least as strong as the weakest link in the Pathsec channel in question. Otherwise, the session is vulernable to downgrade attack. The server and the client must realize that they, individually or jointly, have little control over the activities conducted by trusted intermediaries. Thus frequent audit, or certification if applicable, of trust-worthiness is RECOMMENDED. Each intermediary MUST excercise continuous diligence and self-discipline in securing its own premises in various aspects.

It is possible for "conspiring" intermediaries to modify a routing policy -- e.g. adding or removing hops from a routing metric, practically executing loose source routing instead of strict source routing without the end points' knowledge -- even if the routing metric has been MACed by the server or the client. Some intermediaries that are genuinely trustworthy may find this "feature" a "door" to creative applications, and Pathsec is safe with this "door" unlocked so long as the intermediaries are genuinely trustworthy, albeit occasionally mischievous for their own good. However, there remains the challenge to make this "door" lockable by the server or the client.

6.8 Remote Execute

Semantics for remote execute are not intrinsic to the Pathsec protocol. For example, support for dereferencing a Pathsec node identified as "www.funcity.bom:443/trojanhorse" SHALL NOT be RECOMMENDED. Both the server and the client SHOULD assume that intermediaries are very likely to execute remote procudures at their own discretion. Intermediaries that execute remote procedures MUST adhere to the guidelines set in 6.7.

7 I18N & L10N Considerations

The hopList of a Pathsec Routing Metric is encoded using UTF-8 [UTF8]. Internationalization (I18N) and localization (L10N) should be considered only if future domain names are to be specified in text strings.

8 Intellectual Property Rights

9 Acknowledgments

The author wishes to thank in advance his Digital Island and Cable & Wireless Global colleagues, and the IETF TLS Working Group chair and members for their comments and supports that shall contribute to the advancement of the Pathsec protocol from its current stage.

10 References

[CDI] Content Distribution/Delivery Internetworking
Working Group, IETF.

[DENNING] D. E. Denning, "Cryptography and Data Security,"
Addison-Wesley, 1982.

[HMAC] H. Krawczyk, M. Bellare, and R. Canetti -- HMAC:
Keyed-hashing for message authentication. IETF RFC 2104,
February 1997.

[HTTP] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter,
P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol --
HTTP/1.1," IETF RFC 2616, June 1999.

[IP]   ISI, "Internet Protocol, DARPA Internet Program, Protocol
Specification," IETF RFC 791, September 1981.

[KWORD] S. Bradner, "Key words for use in RFCs to Indicate
Requirement Levels," IETF RFC 2119, March 1997.

[NICHOLS] R.K. Nichols, "ICSA Guide to Cryptography,"
McGraw Hill, 1999.

[OPES] Open Pluggable Edge Services Working Group, IETF.

[RESCORLA] E. Rescorla, "SSL and TLS, Designing and Building
Secure Systems," Addison-Wesley, 2001.

[STARTLS] P. Hoffman, "SMTP Service Extension for Secure SMTP over
TLS,"IETF RFC 2487, January 1999.

[STEVENS] W.R. Stevens, "TCP/IP Illustrated, Vol 1" Addison Wesley,
1994.

[TLS1] T. Dierks, and C. Allen, "The TLS Protocol - Version 1.0,"
IETF RFC 2246, January 1999.

[TLSX] S. Blake-Wilson, M. Nystrom, D. Hopwood, and J. Mikkelsen,
"TLS Extensions, draft-ietf-tls-extensions-00.txt,"
IETF Internet Draft Work-in-progress, June 2001.

[URI] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource
Identifiers (URI): Generic Syntax," IETF RFC 2396, August
1998.

[UTF8] F. Yergeau, "UTF-8, a transformation format of ISO 10646,"

IETF RFC 2279, January 1998.

[WEBI] Web Intermediaries Working Group, IETF.

[XDR] R. Srinivasan, "XDR: External Data Representation Standard," IETF RFC 1832, March 1995.

[XMPR] D.L. Mills, "An Experimental Multiple-Path Routing Algorithm," IETF RFC 981, March 1986.

11 Author's Address

Joseph Hui
Digital Island
a Cable & Wireless company
225 W. Hillcrest Drive
Thousand Oaks, CA 91360
USA

Phone: +1 805 370 2165

Email: jhui@digisle.net