Network Working Group   F. Brockners, Ed.
Internet-Draft   S. Bhandari, Ed.
Intended status: Experimental   Cisco
Expires: November 26, 2020   T. Mizrahi, Ed.
                                          Huawei Network.IO Innovation Lab
                                                               S. Dara
                                                              Seconize
                                                             S. Youell
                                                                  JPMC
                                                          May 25, 2020

                            Proof of Transit
                    draft-ietf-sfc-proof-of-transit-05

Abstract

   Several technologies such as Traffic Engineering (TE), Service
   Function Chaining (SFC), and policy based routing are used to steer
   traffic through a specific, user-defined path. This document defines
   mechanisms to securely prove that traffic transited said defined
   path. These mechanisms allow to securely verify whether, within a
   given path, all packets traversed all the nodes that they are
   supposed to visit.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of [BCP 78](#) and [BCP 79](#).

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on November 26, 2020.

Copyright Notice

Table of Contents

[1](). Introduction

   Several deployments use Traffic Engineering, policy routing, Segment
   Routing (SR), and Service Function Chaining (SFC) [[RFC7665]()] to steer
   packets through a specific set of nodes. In certain cases,
   regulatory obligations or a compliance policy require operators to
   prove that all packets that are supposed to follow a specific path
   are indeed being forwarded across and exact set of pre-determined
   nodes.

   If a packet flow is supposed to go through a series of service
   functions or network nodes, it has to be proven that indeed all
   packets of the flow followed the path or service chain or collection
   of nodes specified by the policy. In case some packets of a flow
   weren't appropriately processed, a verification device should
   determine the policy violation and take corresponding actions
   corresponding to the policy (e.g., drop or redirect the packet, send
   an alert etc.) In today's deployments, the proof that a packet
   traversed a particular path or service chain is typically delivered
   in an indirect way: Service appliances and network forwarding are in
   different trust domains. Physical hand-off-points are defined
   between these trust domains (i.e. physical interfaces). Or in other
   terms, in the "network forwarding domain" things are wired up in a

way that traffic is delivered to the ingress interface of a service
appliance and received back from an egress interface of a service
appliance. This "wiring" is verified and then trusted upon. The
evolution to Network Function Virtualization (NFV) and modern service
chaining concepts (using technologies such as Locator/ID Separation
Protocol (LISP), Network Service Header (NSH), Segment Routing (SR),
etc.) blurs the line between the different trust domains, because the
hand-off-points are no longer clearly defined physical interfaces,
but are virtual interfaces. As a consequence, different trust layers
should not to be mixed in the same device. For an NFV scenario a
different type of proof is required. Offering a proof that a packet
indeed traversed a specific set of service functions or nodes allows

operators to evolve from the above described indirect methods of
proving that packets visit a predetermined set of nodes.

The solution approach presented in this document is based on a small
portion of operational data added to every packet. This "in-situ"
operational data is also referred to as "proof of transit data", or
POT data. The POT data is updated at every required node and is used
to verify whether a packet traversed all required nodes. A
particular set of nodes "to be verified" is either described by a set
of shares of a single secret. Nodes on the path retrieve their
individual shares of the secret using Shamir's Secret Sharing scheme
from a central controller. The complete secret set is only known to
the controller and a verifier node, which is typically the ultimate
node on a path that performs verification. Each node in the path
uses its share of the secret to update the POT data of the packets as
the packets pass through the node. When the verifier receives a
packet, it uses its key along with data found in the packet to
validate whether the packet traversed the path correctly.

2. 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 [RFC2119].

Abbreviations used in this document:

HMAC: Hash based Message Authentication Code. For example,
           HMAC-SHA256 generates 256 bits of MAC

IOAM: In-situ Operations, Administration, and Maintenance

LISP: Locator/ID Separation Protocol

LPC: Lagrange Polynomial Constants

MTU: Maximum Transmit Unit

NFV: Network Function Virtualization

NSH: Network Service Header

POT: Proof of Transit

POT-Profile: Proof of Transit Profile that has the necessary data
           for nodes to participate in proof of transit

RND: Random Bits generated per packet. Packet fields that do
           not change during the traversal are given as input to
           HMAC-256 algorithm. A minimum of 32 bits (left most) need
           to be used from the output if RND is used to verify the
           packet integrity. This is a standard recommendation by
           NIST.

SEQ_NO: Sequence number initialized to a predefined constant.
           This is used in concatenation with RND bits to mitigate
           different attacks discussed later.

SFC: Service Function Chain

SSSS: Shamir's Secret Sharing Scheme

SR: Segment Routing

3. Proof of Transit

This section discusses methods and algorithms to provide for a "proof
of transit" (POT) for packets traversing a specific path. A path
which is to be verified consists of a set of nodes. Transit of the

data packets through those nodes is to be proven. Besides the nodes,
the setup also includes a Controller that creates secrets and secrets
shares and configures the nodes for POT operations.

The methods how traffic is identified and associated to a specific
path is outside the scope of this document. Identification could be
done using a filter (e.g., 5-tuple classifier), or an identifier
which is already present in the packet (e.g., path or service
identifier, NSH Service Path Identifier (SPI), flow-label, etc.)

The POT information is encapsulated in packets as an IOAM Proof of
Transit Option-Type. The details and format of the encapsulation and
the IOAM POT Option-Type format are specified in
[I-D.ietf-ippm-ioam-data].

The solution approach is detailed in two steps. Initially the
concept of the approach is explained. This concept is then further
refined to make it operationally feasible.

3.1. Basic Idea

The method relies on adding POT data to all packets that traverse a
path. The added POT data allows a verifying node (egress node) to
check whether a packet traversed the identified set of nodes on a
path correctly or not. Security mechanisms are natively built into
the generation of the POT data to protect against misuse (e.g.,

configuration mistakes). The mechanism for POT leverages "Shamir's
Secret Sharing" scheme [SSS].

Shamir's secret sharing base idea: A polynomial (represented by its
coefficients) of degree k is chosen as a secret by the controller. A
polynomial represents a curve. A set of k+1 points on the curve
define the polynomial and are thus needed to (re-)construct the
polynomial. Each of these k+1 points of the polynomial is called a
"share" of the secret. A single secret is associated with a
particular set of k+1 nodes, which typically represent the path to be
verified. k+1 shares of the single secret (i.e., k+1 points on the
curve) are securely distributed from a Controller to the network
nodes. Nodes use their respective share to update a cumulative value
in the POT data of each packet. Only a verifying node has access to
the complete secret. The verifying node validates the correctness of

the received POT data by reconstructing the curve.

The polynomial cannot be reconstructed if any of the points are
missed or tampered. Per Shamir's Secret Sharing Scheme, any lesser
points means one or more nodes are missed. Details of the precise
configuration needed for achieving security are discussed further
below.

While applicable in theory, a vanilla approach based on Shamir's
Secret Sharing Scheme could be easily attacked. If the same
polynomial is reused for every packet for a path a passive attacker
could reuse the value. As a consequence, one could consider creating
a different polynomial per packet. Such an approach would be
operationally complex. It would be complex to configure and recycle
so many curves and their respective points for each node. Rather
than using a single polynomial, two polynomials are used for the
solution approach: A secret polynomial as described above which is
kept constant, and a per-packet polynomial which is public and
generated by the ingress node (the first node along the path).
Operations are performed on the sum of those two polynomials –
creating a third polynomial which is secret and per packet.

3.2. Solution Approach

Solution approach: The overall algorithm uses two polynomials: POLY-1
and POLY-2. POLY-1 is secret and constant. A different POLY-1 is
used for each path, and its value is known to the controller and to
the verifier of the respective path. Each node gets a point on
POLY-1 at setup-time and keeps it secret. POLY-2 is public, random
and per packet. Each node generates a point on POLY-2 each time a
packet crosses it. Each node then calculates (point on POLY-1 +
point on POLY-2) to get a (point on POLY-3) and passes it to verifier
by adding it to each packet. The verifier constructs POLY-3 from the

points given by all the nodes and cross checks whether POLY-3 =
POLY-1 + POLY-2. Only the verifier knows POLY-1.

The solution leverages finite field arithmetic in a field of size
"prime number", i.e. all operations are performed "modulo prime
number".

Detailed algorithms are discussed next. A simple example that

describes how the algorithms work is discussed in Section 3.3.

The algorithms themselves do not constrain the ranges of possible
values for the different parameters and coefficients used. A
deployment of the algorithms will always need to define appropriate
ranges. Please refer to the YANG model in Section 5.2 for details on
the units and ranges of possible values of the different parameters
and coefficients.

3.2.1. Setup

A controller generates a first polynomial (POLY-1) of degree k and
k+1 points on the polynomial, corresponding to the k+1 nodes along
the path. The constant coefficient of POLY-1 is considered the
SECRET, which is per the definition of the SSSS algorithm [SSS]. The
k+1 points are used to derive the Lagrange Basis Polynomials. The
Lagrange Polynomial Constants (LPC) are retrieved from the constant
coefficients of the Lagrange Basis Polynomials. Each of the k+1
nodes (including verifier) are assigned a point on the polynomial
i.e., shares of the SECRET. The verifier is configured with the
SECRET. The Controller also generates coefficients (except the
constant coefficient, called "RND", which is changed on a per packet
basis) of a second polynomial POLY-2 of the same degree. Each node
is configured with the LPC of POLY-2. Note that POLY-2 is public.

3.2.2. In Transit

For each packet, the ingress node generates a random number (RND).
It is considered as the constant coefficient for POLY-2. A
cumulative value (CML) is initialized to 0. Both RND, CML are
carried as within the packet POT data. As the packet visits each
node, the RND is retrieved from the packet and the respective share
of POLY-2 is calculated. Each node calculates (Share(POLY-1) +
Share(POLY-2)) and CML is updated with this sum, specifically each
node performs

CML = CML+(((Share(POLY-1)+ Share(POLY-2)) * LPC) mod Prime, with
"LPC" being the Lagrange Polynomial Constant and "Prime" being the
prime number which defines the finite field arithmetic that all

operations are done over. Please also refer to Section 3.3.2 below

for further details how the operations are performed.

This step is performed by each node until the packet completes the path. The verifier also performs the step with its respective share.

3.2.3. Verification

The verifier cross checks whether CML = SECRET + RND. If this matches then the packet traversed the specified set of nodes in the path. This is due to the additive homomorphic property of Shamir's Secret Sharing scheme.

3.3. Illustrative Example

This section shows a simple example to illustrate step by step the approach described above. The example assumes a network with 3 nodes. The last node that packets traverse also serves as the verifier. A Controller communicates the required parameters to the individual nodes.

3.3.1. Baseline

Assumption: It is to be verified whether packets passed through the 3 nodes. A polynomial of degree 2 is chosen for verification.

Choices: Prime = 53. POLY-1(x) = $(3x^2 + 3x + 10)$ mod 53. The secret to be re-constructed is the constant coefficient of POLY-1, i.e., SECRET=10. It is important to note that all operations are done over a finite field (i.e., modulo Prime = 53).

3.3.1.1. Secret Shares

The shares of the secret are the points on POLY-1 chosen for the 3 nodes. For example, let x0=2, x1=4, x2=5.

   POLY-1(2) = 28 => (x0, y0) = (2, 28)

   POLY-1(4) = 17 => (x1, y1) = (4, 17)

   POLY-1(5) = 47 => (x2, y2) = (5, 47)

The three points above are the points on the curve which are considered the shares of the secret. They are assigned by the Controller to three nodes respectively and are kept secret.

3.3.1.2. Lagrange Polynomials

   Lagrange basis polynomials (or Lagrange polynomials) are used for
   polynomial interpolation. For a given set of points on the curve
   Lagrange polynomials (as defined below) are used to reconstruct the
   curve and thus reconstruct the complete secret.

      l0(x) = (((x-x1) / (x0-x1)) * ((x-x2)/x0-x2))) mod 53
            = (((x-4) / (2-4)) * ((x-5)/2-5))) mod 53
            = (10/3 - 3x/2 + (1/6)x^2) mod 53

      l1(x) = (((x-x0) / (x1-x0)) * ((x-x2)/x1-x2))) mod 53
            = (-5 + 7x/2 - (1/2)x^2) mod 53

      l2(x) = (((x-x0) / (x2-x0)) * ((x-x1)/x2-x1))) mod 53
            = (8/3 - 2 + (1/3)x^2) mod 53

3.3.1.3. LPC Computation

   Since x0=2, x1=4, x2=5 are chosen points. Given that computations
   are done over a finite arithmetic field ("modulo a prime number"),
   the Lagrange basis polynomial constants are computed modulo 53. The
   Lagrange Polynomial Constants (LPC) would be mod(10/3, 53), mod(-5,
   53), mod(8/3, 53).LPC are computed by the Controller and communicated
   to the individual nodes.

      LPC(l0) = (10/3) mod 53 = 21

      LPC(l1) = (-5) mod 53 = 48

      LPC(l2) = (8/3) mod 53 = 38

   For a general way to compute the modular multiplicative inverse, see
   e.g., the Euclidean algorithm.

3.3.1.4. Reconstruction

   Reconstruction of the polynomial is well-defined as

   POLY1(x) = l0(x) * y0 + l1(x) * y1 + l2(x) * y2

   Subsequently, the SECRET, which is the constant coefficient of
   POLY1(x) can be computed as below

```
     SECRET = (y0*LPC(l0)+y1*LPC(l1)+y2*LPC(l2)) mod 53
```

The secret can be easily reconstructed using the y-values and the
LPC:

```
SECRET = (y0*LPC(l0) + y1*LPC(l1) + y2*LPC(l2)) mod 53
       = (28 * 21 + 17 * 48 + 47 * 38) mod 53
       = 3190 mod 53
       = 10
```

One observes that the secret reconstruction can easily be performed
cumulatively hop by hop, i.e. by every node. CML represents the
cumulative value. It is the POT data in the packet that is updated
at each hop with the node's respective (yi*LPC(i)), where i is their
respective value.

3.3.1.5. Verification

Upon completion of the path, the resulting CML is retrieved by the
verifier from the packet POT data. Recall that the verifier is
preconfigured with the original SECRET. It is cross checked with the
CML by the verifier. Subsequent actions based on the verification
failing or succeeding could be taken as per the configured policies.

3.3.2. Complete Solution

As observed previously, the baseline algorithm that involves a single
secret polynomial is not secure. The complete solution leverages a
random second polynomial, which is chosen per packet.

3.3.2.1. Random Polynomial

Let the second polynomial POLY-2 be (RND + 7x + 10 x^2). RND is a
random number and is generated for each packet. Note that POLY-2 is
public and need not be kept secret. The nodes can be pre-configured
with the non-constant coefficients (for example, 7 and 10 in this
case could be configured through the Controller on each node). So
precisely only the RND value changes per packet and is public and the
rest of the non-constant coefficients of POLY-2 is kept secret.

3.3.2.2. Reconstruction

Recall that each node is preconfigured with their respective
Share(POLY-1). Each node calculates its respective Share(POLY-2)
using the RND value retrieved from the packet. The CML
reconstruction is enhanced as below. At every node, CML is updated
as

CML = CML+(((Share(POLY-1)+ Share(POLY-2)) * LPC) mod Prime

Let us observe the packet level transformations in detail. For the
example packet here, let the value RND be 45. Thus POLY-2 would be
$(45 + 7x + 10x^2)$.

The shares that could be generated are (2, 46), (4, 21), (5, 12).

   At ingress: The fields RND = 45. CML = 0.

   At node-1 (x0): Respective share of POLY-2 is generated i.e., (2,
   46) because share index of node-1 is 2.

   CML = 0 + ((28 + 46)* 21) mod 53 = 17

   At node-2 (x1): Respective share of POLY-2 is generated i.e., (4,
   21) because share index of node-2 is 4.

   CML = 17 + ((17 + 21)*48) mod 53 = 17 + 22 = 39

   At node-3 (x2), which is also the verifier: The respective share
   of POLY-2 is generated i.e., (5, 12) because the share index of
   the verifier is 12.

   CML = 39 + ((47 + 12)*38) mod 53 = 39 + 16 = 55 mod 53 = 2

The verification using CML is discussed in next section.

3.3.2.3. Verification

As shown in the above example, for final verification, the verifier
compares:

VERIFY = (SECRET + RND) mod Prime, with Prime = 53 here

VERIFY = (RND-1 + RND-2) mod Prime = (10 + 45) mod 53 = 2

    Since VERIFY = CML the packet is proven to have gone through nodes 1,
    2, and 3.

3.3.3. Solution Deployment Considerations

    The "complete solution" described above in Section 3.3.2 could still
    be prone to replay or preplay attacks. An attacker could e.g. reuse
    the POT metadata for bypassing the verification. These threats can
    be mitigated by appropriate parameterization of the algorithm.
    Please refer to Section 7 for details.

3.4. Operational Aspects

    To operationalize this scheme, a central controller is used to
    generate the necessary polynomials, the secret share per node, the
    prime number, etc. and distributing the data to the nodes
    participating in proof of transit. The identified node that performs
    the verification is provided with the verification key. The
    information provided from the Controller to each of the nodes
    participating in proof of transit is referred to as a proof of
    transit profile (POT-Profile). Also note that the set of nodes for
    which the transit has to be proven are typically associated to a
    different trust domain than the verifier. Note that building the
    trust relationship between the Controller and the nodes is outside
    the scope of this document. Techniques such as those described in
    [I-D.ietf-anima-autonomic-control-plane] might be applied.

    To optimize the overall data amount of exchanged and the processing
    at the nodes the following optimizations are performed:

    1. The points (x, y) for each of the nodes on the public and private
        polynomials are picked such that the x component of the points
        match. This lends to the LPC values which are used to calculate
        the cumulative value CML to be constant. Note that the LPC are
        only depending on the x components. They can be computed at the
        controller and communicated to the nodes. Otherwise, one would

need to distributed the x components to all the nodes.

2. A pre-evaluated portion of the public polynomial for each of the
    nodes is calculated and added to the POT-Profile. Without this
    all the coefficients of the public polynomial had to be added to
    the POT profile and each node had to evaluate them. As stated
    before, the public portion is only the constant coefficient RND
    value, the pre-evaluated portion for each node should be kept
    secret as well.

3. To provide flexibility on the size of the cumulative and random
    numbers carried in the POT data a field to indicate this is
    shared and interpreted at the nodes.

[3.5](). Ordered POT (OPOT)

POT as discussed in this document so far only verifies that a defined
set of nodes have been traversed by a packet. The order in which
nodes where traversed is not verified. "Ordered Proof of Transit
(OPOT)" addresses the need of deployments, that require to verify the
order in which nodes were traversed. OPOT extends the POT scheme
with symmetric masking between the nodes.

1. For each path the controller provisions all the nodes with (or
    asks them to agree on) two secrets per node, that we will refer
    to as masks, one for the connection from the upstream node(s),
    another for the connection to the downstream node(s). For
    obvious reasons, the ingress and egress (verifier) nodes only
    receive one, for downstream and upstream, respectively.

2. Any two contiguous nodes in the OPOT stream share the mask for
    the connection between them, in the shape of symmetric keys.
    Masks can be refreshed as per-policy, defined at each hop or
    globally by the controller.

3. Each mask has the same size in bits as the length assigned to CML
    plus RND, as described in the above sections.

4. Whenever a packet is received at an intermediate node, the
    CML+RND sequence is deciphered (by XORing, though other ciphering
    schemas MAY be possible) with the upstream mask before applying

the procedures described in [Section 3.3.2](#).

5. Once the new values of CML+RND are produced, they are ciphered
    (by XORing, though other ciphering schemas MAY be possible) with
    the downstream mask before transmitting the packet to the next
    node downstream.

6. The ingress node only applies step 5 above, while the verifier
    only applies step 4 before running the verification procedure.

The described process allows the verifier to check if the packet has
followed the correct order while traversing the path. In particular,
the reconstruction process will fail if the order is not respected,
as the deciphering process will produce invalid CML and RND values,
and the interpolation (secret reconstruction) will finally generate a
wrong verification value.

This procedure does not impose a high computational burden, does not
require additional packet overhead, can be deployed on chains of any
length, does not require any node to be aware of any additional
information than the upstream and downstream masks, and can be
integrated with the other operational mechanisms applied by the
controller to distribute shares and other secret material.

4. Sizing the Data for Proof of Transit

Proof of transit requires transport of two data fields in every
packet that should be verified:

1. RND: Random number (the constant coefficient of public
    polynomial)

2. CML: Cumulative

The size of the data fields determines how often a new set of
polynomials would need to be created. At maximum, the largest RND
number that can be represented with a given number of bits determines
the number of unique polynomials POLY-2 that can be created. The
table below shows the maximum interval for how long a single set of
polynomials could last for a variety of bit rates and RND sizes: When

choosing 64 bits for RND and CML data fields, the time between a
renewal of secrets could be as long as 3,100 years, even when running
at 100 Gbps.

Transfer rate	Secret/RND size	Max # of packets	Time RND lasts
1 Gbps	64	2^{64} = approx. $2*10^{19}$	approx. 310,000 years
10 Gbps	64	2^{64} = approx. $2*10^{19}$	approx. 31,000 years
100 Gbps	64	2^{64} = approx. $2*10^{19}$	approx. 3,100 years
1 Gbps	32	2^{32} = approx. $4*10^{9}$	2,200 seconds
10 Gbps	32	2^{32} = approx. $4*10^{9}$	220 seconds
100 Gbps	32	2^{32} = approx. $4*10^{9}$	22 seconds

Table assumes 64 octet packets

Table 1: Proof of transit data sizing

If the symmetric masking method for ordered POT is used
(Section 3.5), the masks used between nodes adjacent in the path MUST
have a length equal to the sum of the ones of RND and CML.

5. Node Configuration

A POT system consists of a number of nodes that participate in POT
and a Controller, which serves as a control and configuration entity.
The Controller is to create the required parameters (polynomials,
prime number, etc.) and communicate the associated values (i.e. prime
number, secret-share, LPC, etc.) to the nodes. The sum of all

parameters for a specific node is referred to as "POT-Profile". For
details see the YANG model in Section 5.2.This document does not
define a specific protocol to be used between Controller and nodes.
It only defines the procedures and the associated YANG data model.

5.1. Procedure

   The Controller creates new POT-Profiles at a constant rate and
   communicates the POT-Profile to the nodes. The controller labels a
   POT-Profile "even" or "odd" and the Controller cycles between "even"
   and "odd" labeled profiles. This means that the parameters for the
   algorithms are continuously refreshed. Please refer to Section 4 for
   choosing an appropriate refresh rate: The rate at which the POT-
   Profiles are communicated to the nodes is configurable and MUST be
   more frequent than the speed at which a POT-Profile is "used up".
   Once the POT-Profile has been successfully communicated to all nodes
   (e.g., all NETCONF transactions completed, in case NETCONF is used as
   a protocol), the controller sends an "enable POT-Profile" request to
   the ingress node.

   All nodes maintain two POT-Profiles (an even and an odd POT-Profile):
   One POT-Profile is currently active and in use; one profile is
   standby and about to get used. A flag in the packet is indicating
   whether the odd or even POT-Profile is to be used by a node. This is
   to ensure that during profile change the service is not disrupted.
   If the "odd" profile is active, the Controller can communicate the
   "even" profile to all nodes. Only if all the nodes have received the
   POT-Profile, the Controller will tell the ingress node to switch to
   the "even" profile. Given that the indicator travels within the
   packet, all nodes will switch to the "even" profile. The "even"
   profile gets active on all nodes and nodes are ready to receive a new
   "odd" profile.

   Unless the ingress node receives a request to switch profiles, it'll
   continue to use the active profile. If a profile is "used up" the
   ingress node will recycle the active profile and start over (this
   could give rise to replay attacks in theory - but with 2^32 or 2^64
   packets this isn't really likely in reality).

5.2. YANG Model for POT

   This section defines that YANG data model for the information
   exchange between the Controller and the node.

5.2.1. Main Parameters

   The main parameters for the information exchange between the
   Controller and the node used in the YANG model are as follows:

   o pot-profile-index: Section 5.1 details that two POT-Profiles are
      used. Only one of the POT-Profiles is active at a given point in
      time, allowing the Controller to refresh the non-active one for
      future use. pot-profile-index defines which of the POT-Profiles
      (the "even" or "odd" POT-Profile) is currently active. pot-
      profile-index will be set in the first hop of the path or chain.
      Other nodes will not use this field.

   o prime-number: Prime number used for module math computation.

   o secret-share: Share of the secret of polynomial-1 used in
      computation for the node. If POLY-1 is defined by points $(x1_i,
      y1_i)$ with $i=0,..k$, then for node i, the secret-share will be
      $y1\_i$.

   o public-polynomial: Public polynomial value for the node.. If
      POLY-2 is defined by points $(x2\_i, y2\_i)$ with $i=0,..k$, then for
      node i, the secret-share will be $y2\_i$.

   o lpc: Lagrange Polynomial Coefficient for the node, i.e. for node
      i, this would be $LPC(l\_i)$, with l_i being the i-th Lagrange Basis
      Polynomial.

   o validator?: True if the node is a verifier node.

   o validator-key?: The validator-key represents the SECRET as
      described in the sections above. The SECRET is the constant
      coefficient of POLY-1(z). If $POLY-1(z) = a_0 + a_1*z +
      a_2*z^2+..+a_k*z^k$, then the SECRET would be a_0.

   o bitmask?: Number of bits as mask used in controlling the size of
      the random value generation. 32-bits of mask is default. See
      Section 4 for details.

5.2.2. Tree Diagram

   This section shows a simplified graphical representation of the YANG
   data model for POT. The meaning of the symbols in these diagrams is
   as follows:

   o Brackets "[" and "]" enclose list keys.

   o Abbreviations before data node names: "rw" means configuration
      (read-write), and "ro" means state data (read-only).

   o Symbols after data node names: "?" means an optional node, "!"
      means a presence container, and "*" denotes a list and leaf-list.

   o Parentheses enclose choice and case nodes, and case nodes are also
      marked with a colon (":").

   o Ellipsis ("...") stands for contents of subtrees that are not
      shown.

```
   <CODE BEGINS>
   module: ietf-pot-profile
     +--rw pot-profiles
        +--rw pot-profile-set* [pot-profile-name]
           +--rw pot-profile-name        string
           +--rw active-profile-index?   profile-index-range
           +--rw pot-profile-list* [pot-profile-index]
              +--rw pot-profile-index    profile-index-range
              +--rw prime-number         uint64
              +--rw secret-share         uint64
              +--rw public-polynomial    uint64
              +--rw lpc                  uint64
              +--rw validator?           boolean
              +--rw validator-key?       uint64
              +--rw bitmask?             uint64
              +--rw opot-masks
                 +--rw downstream-mask*   uint64
                 +--rw upstream-mask*     uint64
   <CODE ENDS>
```

5.2.3. YANG Model

```
   <CODE BEGINS> file "ietf-pot-profile@2016-06-15.yang"
   module ietf-pot-profile {

     yang-version 1;

     namespace "urn:ietf:params:xml:ns:yang:ietf-pot-profile";
```

```
    prefix ietf-pot-profile;

    organization "IETF SFC Working Group";

    contact "WG Web:    <https://tools.ietf.org/wg/sfc/>
             WG List:  <mailto:sfc@ietf.org>";
```

```
    description
       "This module contains a collection of YANG
        definitions for proof of transit configuration
        parameters. The model is meant for proof of
        transit and is targeted for communicating the
        POT-Profile between a controller and nodes
        participating in proof of transit.

        Copyright (c) 2018 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.
        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (http://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX; see
        the RFC itself for full legal notices.

        Copyright (c) 2018 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

        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 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.";

    revision 2016-06-15 {
      description
        "Initial revision.";
      reference
```

```
        "";
      }

      typedef profile-index-range {
        type int32 {
          range "0 .. 1";
        }
        description
          "Range used for the profile index. Currently restricted to
           0 or 1 to identify the odd or even profiles.";
      }


      grouping pot-profile {
        description "A grouping for proof of transit profiles.";
```

```
        list pot-profile-list {
          key "pot-profile-index";
          ordered-by user;
          description "A set of pot profiles.";

          leaf pot-profile-index {
            type profile-index-range;
            mandatory true;
            description
              "Proof of transit profile index.";
          }

          leaf prime-number {
            type uint64;
            mandatory true;
            description
              "Prime number used for module math computation";
          }

          leaf secret-share {
            type uint64;
            mandatory true;
            description
              "Share of the secret of polynomial-1 used
               in computation for the node. If POLY-1
               is defined by points (x1_i, y1_i) with
```

```
            i=0,..k, then for node i, the secret-share
            will be y1_i.";
        }

        leaf public-polynomial {
          type uint64;
          mandatory true;
          description
            "Public polynomial value for the node.
             If POLY-2 is defined by points (x2_i, y2_i)
             with i=0,..k, then for node i,
             the secret-share will be y2_i.";
        }

        leaf lpc {
          type uint64;
          mandatory true;
          description
            "Lagrange Polynomial Coefficient";
        }

        leaf validator {
```

```
          type boolean;
          default "false";
          description
            "True if the node is a verifier node";
        }

        leaf validator-key {
          type uint64;
          description
            "The validator-key represents the secret.
             The secret is the constant coefficient of
             POLY-1(z). If POLY-1(z) =
             a_0 + a_1*z + a_2*z^2+..+a_k*z^k,
             then the SECRET would be a_0.";
        }

        leaf bitmask {
          type uint64;
          default 4294967295;
```

```
            description
              "Number of bits as mask used in controlling
               the size of the random value generation.
               32-bits of mask is default.";
          }

          uses opot-profile;

        }
      }

      grouping opot-profile {
       description "Grouping containing OPoT related data.";


        container opot-masks {
          must "count(downstream-mask) = count(upstream-mask)";
          description "Masking information for OPoT support.";

          leaf-list downstream-mask {
            type uint64;
            max-elements 2;
            description "Secret stream used to demask the PoT metadata.
            The mask is used between nodes adjacent in the path
            and MUST have a length equal to the sum of the ones
            of RND and CML.";
          }

          leaf-list upstream-mask {
```

```
            type uint64;
            max-elements 2;
            description "Secret stream used to mask the PoT metadata.
            The mask is used between nodes adjacent in the path
            and MUST have a length equal to the sum of the ones
            of RND and CML.";
          }
        }
      }

      container pot-profiles {
        description "A group of proof of transit profiles.";
```

```
      list pot-profile-set {
        key "pot-profile-name";
        ordered-by user;
        description
          "Set of proof of transit profiles that group parameters
           required to classify and compute proof of transit
           metadata at a node";

        leaf pot-profile-name {
          type string;
          mandatory true;
          description
            "Unique identifier for each proof of transit profile";
        }

        leaf active-profile-index {
          type profile-index-range;
          description
            "POT-Profile index that is currently active.
             Will be set in the first hop of the path or chain.
             Other nodes will not use this field.";
        }

        uses pot-profile;
      }
    /*** Container: end ***/
    }
  /*** module: end ***/
  }
  <CODE ENDS>
```

6. IANA Considerations

   This document does not require any actions from IANA.

7. Security Considerations

POT is a mechanism that is used for verifying the path through which
a packet was forwarded. The security considerations of IOAM in
general are discussed in [I-D.ietf-ippm-ioam-data]. Specifically, it
is assumed that POT is used in a confined network domain, and
therefore the potential threats that POT is intended to mitigate
should be viewed accordingly. POT prevents spoofing and tampering;
an attacker cannot maliciously create a bogus POT or modify a
legitimate one. Furthermore, a legitimate node that takes part in
the POT protocol cannot masquerade as another node along the path.
These considerations are discussed in detail in the rest of this
section.

7.1. Proof of Transit

Proof of correctness and security of the solution approach is per
Shamir's Secret Sharing Scheme [SSS]. Cryptographically speaking it
achieves information-theoretic security i.e., it cannot be broken by
an attacker even with unlimited computing power. As long as the
below conditions are met it is impossible for an attacker to bypass
one or multiple nodes without getting caught.

o If there are k+1 nodes in the path, the polynomials (POLY-1, POLY-
   2) should be of degree k. Also k+1 points of POLY-1 are chosen
   and assigned to each node respectively. The verifier can re-
   construct the k degree polynomial (POLY-3) only when all the
   points are correctly retrieved.

o Precisely three values are kept secret by individual nodes. Share
   of SECRET (i.e. points on POLY-1), Share of POLY-2, LPC, P. Note
   that only constant coefficient, RND, of POLY-2 is public. x values
   and non-constant coefficient of POLY-2 are secret

An attacker bypassing a few nodes will miss adding a respective point
on POLY-1 to corresponding point on POLY-2 , thus the verifier cannot
construct POLY-3 for cross verification.

Also it is highly recommended that different polynomials should be
used as POLY-1 across different paths, traffic profiles or service
chains.

If symmetric masking is used to assure OPOT (Section 3.5), the nodes
need to keep two additional secrets: the downstream and upstream

masks, that have to be managed under the same conditions as the
secrets mentioned above. And it is equally recommended to employ a
different set of mask pairs across different paths, traffic profiles
or service chains.

7.2. Cryptanalysis

A passive attacker could try to harvest the POT data (i.e., CML, RND
values) in order to determine the configured secrets. Subsequently
two types of differential analysis for guessing the secrets could be
done.

o Inter-Node: A passive attacker observing CML values across nodes
   (i.e., as the packets entering and leaving), cannot perform
   differential analysis to construct the points on POLY-1. This is
   because at each point there are four unknowns (i.e. Share(POLY-
   1), Share(Poly-2) LPC and prime number P) and three known values
   (i.e. RND, CML-before, CML-after). The application of symmetric
   masking for OPOT makes inter-node analysis less feasible.

o Inter-Packets: A passive attacker could observe CML values across
   packets (i.e., values of PKT-1 and subsequent PKT-2), in order to
   predict the secrets. Differential analysis across packets could
   be mitigated using a good PRNG for generating RND. Note that if
   constant coefficient is a sequence number than CML values become
   quite predictable and the scheme would be broken. If symmetric
   masking is used for OPOT, inter-packet analysis could be applied
   to guess mask values, which requires a proper refresh rate for
   masks, at least as high as the one used for LPCs.

7.3. Anti-Replay

A passive attacker could reuse a set of older RND and the
intermediate CML values. Thus, an attacker can attack an old
(replayed) RND and CML with a new packet in order to bypass some of
the nodes along the path.

Such attacks could be avoided by carefully choosing POLY-2 as a
(SEQ_NO + RND). For example, if 64 bits are being used for POLY-2
then first 16 bits could be a sequence number SEQ_NO and next 48 bits
could be a random number.

Subsequently, the verifier could use the SEQ_NO bits to run classic
anti-replay techniques like sliding window used in IPSEC. The
verifier could buffer up to 2^{16} packets as a sliding window.
Packets arriving with a higher SEQ_NO than current buffer could be
flagged legitimate. Packets arriving with a lower SEQ_NO than
current buffer could be flagged as suspicious.

For all practical purposes in the rest of the document RND means
SEQ_NO + RND to keep it simple.

The solution discussed in this memo does not currently mitigate
replay attacks. An anti-replay mechanism may be included in future
versions of the solution.

7.4. Anti-Preplay

An active attacker could try to perform a man-in-the-middle (MITM)
attack by extracting the POT of PKT-1 and using it in PKT-2.
Subsequently attacker drops the PKT-1 in order to avoid duplicate POT
values reaching the verifier. If the PKT-1 reaches the verifier,
then this attack is same as Replay attacks discussed before.

Preplay attacks are possible since the POT metadata is not dependent
on the packet fields. Below steps are recommended for remediation:

o Ingress node and Verifier are configured with common pre shared
   key

o Ingress node generates a Message Authentication Code (MAC) from
   packet fields using standard HMAC algorithm.

o The left most bits of the output are truncated to desired length
   to generate RND. It is recommended to use a minimum of 32 bits.

o The verifier regenerates the HMAC from the packet fields and
   compares with RND. To ensure the POT data is in fact that of the
   packet.

If an HMAC is used, an active attacker lacks the knowledge of the
pre-shared key, and thus cannot launch preplay attacks.

The solution discussed in this memo does not currently mitigate
preplay attacks. A mitigation mechanism may be included in future
versions of the solution.

7.5. Tampering

An active attacker could not insert any arbitrary value for CML.
This would subsequently fail the reconstruction of the POLY-3. Also

an attacker could not update the CML with a previously observed
value. This could subsequently be detected by using timestamps
within the RND value as discussed above.

7.6. Recycling

The solution approach is flexible for recycling long term secrets
like POLY-1. All the nodes could be periodically updated with shares
of new SECRET as best practice. The table above could be consulted
for refresh cycles (see Section 4).

If symmetric masking is used for OPOT (Section 3.5), mask values must
be periodically updated as well, at least as frequently as the other
secrets are.

7.7. Redundant Nodes and Failover

A "node" or "service" in terms of POT can be implemented by one or
multiple physical entities. In case of multiple physical entities
(e.g., for load-balancing, or business continuity situations -
consider for example a set of firewalls), all physical entities which
are implementing the same POT node are given that same share of the
secret. This makes multiple physical entities represent the same POT
node from an algorithm perspective.

7.8. Controller Operation

The Controller needs to be secured given that it creates and holds
the secrets, as need to be the nodes. The communication between
Controller and the nodes also needs to be secured. As secure
communication protocol such as for example NETCONF over SSH should be
chosen for Controller to node communication.

The Controller only interacts with the nodes during the initial
configuration and thereafter at regular intervals at which the
operator chooses to switch to a new set of secrets. In case 64 bits
are used for the data fields "CML" and "RND" which are carried within
the data packet, the regular intervals are expected to be quite long
(e.g., at 100 Gbps, a profile would only be used up after 3100 years)

- see [Section 4](#) above, thus even a "headless" operation without a
    Controller can be considered feasible. In such a case, the
    Controller would only be used for the initial configuration of the
    POT-Profiles.

    If OPOT ([Section 3.5](#)) is applied using symmetric masking, the
    Controller will be required to perform a a periodic refresh of the
    mask pairs. The use of OPOT SHOULD be configurable as part of the
    required level of assurance through the Controller management
    interface.

7.9. Verification Scope

    The POT solution defined in this document verifies that a data-packet
    traversed or transited a specific set of nodes. From an algorithm
    perspective, a "node" is an abstract entity. It could be represented
    by one or multiple physical or virtual network devices, or is could
    be a component within a networking device or system. The latter
    would be the case if a forwarding path within a device would need to
    be securely verified.

7.9.1. Node Ordering

    POT using Shamir's secret sharing scheme as discussed in this
    document provides for a means to verify that a set of nodes has been
    visited by a data packet. It does not verify the order in which the
    data packet visited the nodes.

    In case the order in which a data packet traversed a particular set
    of nodes needs to be verified as well, the alternate schemes related
    to OPOT ([Section 3.5](#)) have to be considered. Since these schemes
    introduce at least additional control requirements, the selection of
    order verification SHOULD be configurable the Controller management
    interface.

7.9.2. Stealth Nodes

    The POT approach discussed in this document is to prove that a data
    packet traversed a specific set of "nodes". This set could be all

nodes within a path, but could also be a subset of nodes in a path.
Consequently, the POT approach isn't suited to detect whether
"stealth" nodes which do not participate in proof-of-transit have
been inserted into a path.

8. Acknowledgements

The authors would like to thank Eric Vyncke, Nalini Elkins, Srihari
Raghavan, Ranganathan T S, Karthik Babu Harichandra Babu, Akshaya
Nadahalli, Erik Nordmark, and Andrew Yourtchenko for the comments and
advice.

9. Contributors

In addition to editors and authors listed on the title page, the
following people have contributed substantially to this document and
should be considered coauthors:

      Carlos Pignataro
      Cisco Systems, Inc.
      7200-11 Kit Creek Road
      Research Triangle Park, NC 27709
      United States
      Email: cpignata@cisco.com

      John Leddy
      Email: john@leddy.net

      David Mozes
      Email: mosesster@gmail.com

      Alejandro Aguado
      Universidad Politecnica de Madrid
      Campus Montegancedo, Boadilla del Monte
      Madrid 28660
      Spain

         Phone: +34 910 673 086
         Email: a.aguadom@fi.upm.es

         Diego R. Lopez
         Telefonica I+D
         Editor Jose Manuel Lara, 9 (1-B)
         Seville 41013
         Spain
         Phone: +34 913 129 041
         Email: diego.r.lopez@telefonica.com

10. References

10.1. Normative References

   [I-D.ietf-ippm-ioam-data]
               Brockners, F., Bhandari, S., Pignataro, C., Gredler, H.,
               Leddy, J., Youell, S., Mizrahi, T., Mozes, D., Lapukhov,
               P., remy@barefootnetworks.com, r., daniel.bernier@bell.ca,
               d., and J. Lemon, "Data Fields for In-situ OAM", draft-
               ietf-ippm-ioam-data-09 (work in progress), March 2020.

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
               Requirement Levels", BCP 14, RFC 2119,
               DOI 10.17487/RFC2119, March 1997,
               <https://www.rfc-editor.org/info/rfc2119>.

   [RFC7665]   Halpern, J., Ed. and C. Pignataro, Ed., "Service Function
               Chaining (SFC) Architecture", RFC 7665,
               DOI 10.17487/RFC7665, October 2015,
               <https://www.rfc-editor.org/info/rfc7665>.

   [SSS]   "Shamir's Secret Sharing",
               <https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing>.

10.2. Informative References

    [I-D.ietf-anima-autonomic-control-plane]
              Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic
              Control Plane (ACP)", draft-ietf-anima-autonomic-control-
              plane-18 (work in progress), August 2018.

Authors' Addresses

    Frank Brockners (editor)
    Cisco Systems, Inc.
    Hansaallee 249, 3rd Floor
    DUESSELDORF, NORDRHEIN-WESTFALEN 40549
    Germany

    Email: fbrockne@cisco.com

    Shwetha Bhandari (editor)
    Cisco Systems, Inc.
    Cessna Business Park, Sarjapura Marathalli Outer Ring Road
    Bangalore, KARNATAKA 560 087
    India

    Email: shwethab@cisco.com

    Tal Mizrahi (editor)
    Huawei Network.IO Innovation Lab
    Israel

    Email: tal.mizrahi.phd@gmail.com

    Sashank Dara
    Seconize
    BANGALORE, Bangalore, KARNATAKA
    INDIA

    Email: sashank@seconize.co

Stephen Youell
JP Morgan Chase
25 Bank Street
London   E14 5JP
United Kingdom

Email: stephen.youell@jpmorgan.com