The ESP Stream Transform
draft-caronni-esp-stream-00.txt

Status of this Memo
---------------
This document is an Internet-Draft. Internet Drafts are working docu-
ments of the Internet Engineering Task Force (IETF), its areas, and its
working Groups. Note that other groups may also distribute working docu-
ments as Internet Drafts.

Internet-Drafts draft documents are 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."

To learn the current status of any Internet-Draft, please check the
"id-abstracts.txt" listing contained in the Internet-Drafts Shadow
Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe),
munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or
ftp.isi.edu (US West Coast).

Distribution of this memo is unlimited.

The authors intend to publish this draft as an Experimental RFC after it
has been reviewed within the IPsec Working Group.

Abstract
-------
This document describes a security transform providing privacy and
(optional) replay protection through Encapsulating Secure Payload (ESP).
The transform defines how to use ESP in conjunction with byte-oriented
stream ciphers, such as RC4 or SEAL. These stream ciphers offer strong
encryption with comparatively low computational demands, and are thus
favorable for multimedia bulk data or environments where the computing
power needed per packet should be low.
Table of Contents
-------------
1. Introduction
1.1. Conventions
1.2. Cipher Overview
2. Payload Format
2.1. Stream Offset
2.2. Replay Protection Using the Stream Offset
2.3. Authentication Issues
3. Encryption
4. Decryption
5. Security Considerations
6. Implementation Details
6.1. Summary of Limits
6.2. Technical Notes
7. Acknowledgements
8. References
9. Author addresses

1. Introduction
-------------
The mechanisms presented in this document provide confidentiality and optional replay protection for IP datagrams, using the Encapsulating Security Payload (ESP) [RFC1827] format. Although ESP may also provide data integrity and authenticity, `esp-stream` only offers encryption.

This document assumes that the reader is familiar with the related document "Security Architecture for the Internet Protocol" [RFC1825], which defines the overall security plan for IP and provides important background. Additionally, familiarity with "IP Encapsulating Security Payload (ESP)" [RFC1827] is required, as some of the mechanisms employed here are described in greater depth in that document.

1.1. Conventions
-------------
There are a few conventions we followed in this document what the reader should be aware of:

a) Whenever it is said that something is `"LIMITED"`, the appropriate maximum and minimum values plus a set of recommended values are given in section 6.1.

b) The meanings of capitalized `"MUST"`, `"MUST NOT"`, `"SHOULD"`, and `"MAY"` apply as declared in [RFC1825].

c) Variables and constants in pseudo code are in ALL CAPITALS, to clearly distinguish actual code symbols from the description of the action.
1.2. Cipher Overview
---------------------

Fast algorithms, such as RC4 (devised by Ron Rivest) or SEAL (by Phil Rogaway and Don Coppersmith) are very efficient and easily understandable symmetric additive stream ciphers with a large internal state. The efficiency with which they operate make them ideal candidates for encrypting/decrypting high-throughput, low-latency data streams in software with relatively low CPU usage or just whenever low CPU overhead for security is needed. Such data streams are very common in modern multi-media applications who transmit video data and/or high quality audio data. RC4 has been used (together with SKIP prototypes) to encrypt interactive video connections over IP. Resulting quality was adequate and acceptable.

For all stream ciphers, key lengths of up to and including 128 bits MUST be supported by the implementation, although any particular key may be shorter. Longer keys are strongly recommended.

RC4 allows for key sizes of up to 256 bytes. The key is used to permute the values of 0 to 255 in an array of 256 entries, thus creating the internal state for the pseudo random generator, whose output is used as a one time pad for encryption/decryption. The initial state after keying can thus be one out of 256! states, being equivalent to about 1676 bits key state.

SEAL supports position-dependent key setup, expects 160 bits of key material and allows initializing at any 32 bit offset into the stream. SEAL may in certain environments be even faster than RC4. Key setup in SEAL is much more expensive, as SHA is employed to generate the internal state.

For more details about RC4, SEAL, and about using other stream ciphers, refer to [Schn96]. Please note that SEAL is being patented by IBM, and RC4, although interoperable code is available, is a proprietary algorithm of RSA Data Security Inc. RC4 is trademarked by RSA DSI, anybody contemplating its use should contact them.

2. Payload Format
------------------

The following diagram describes the basic format of IP packets containing the 'esp-stream' header and data. The selection of the actual stream cipher and the key to be used has to take place on a upper layer.

```
|<---- Unencrypted ---->|<---- Encrypted ---->|<---- Type ---->|
+-----------------------+-----------------------+-+-
| IP Header | IP Ext. Headers | ESP Header | Encrypted Data |
+-----------------------+-----------------------+-+-
```

Caronni, Waldvogel
A more detailed diagram of the format of the particular ESP Header (labelled "E" in the left column shown in diagram below) to be used with stream ciphers and the data following it (Data labelled "D", Type labelled "T") is given below, together with a short description of every field.

```
|1 2 3 4 5 6 7 8|1 2 3 4 5 6 7 8|1 2 3 4 5 6 7 8|1 2 3 4 5 6 7 8|
- +---------------+---------------+---------------+---------------+
E |                Security Parameters Index (SPI)                |
E +---------------------------------------------------------------+
E |                                                               |
E |                    Stream Offset (64 bits)                    |
E |                                                               |
- +---------------------------------------------------------------+
D |                                                               |
D :       Encrypted Payload Data (arbitrary number of bytes)      :
D |                                                               |
- +---------------------------------------------------------------+
T |                               +---------------+---------------+
   | Payload Type  |
   +-------------------------------+---------------+
```

Notes
-----

Security Parameters Index (SPI)
A 32-bit value identifying the Security Parameters for this datagram (network order). The value MUST NOT be zero.

Stream Offset (64 bits)
Indicates the number of data bytes already encrypted under the current key for this cipher before this packet was encrypted. This value is stored in network order (most significant octet first) [RFC1700]. For details, see section 2.1.

Encrypted Payload Data (arbitrary number of bytes)
Encapsulated IP or other protocol data, encrypted. No padding is required.

Payload Type (8 bits)
Contains the encrypted type of the next protocol header in the encrypted payload data. Up-to-date values of the IP Protocol/Payload are specified in the most recent "Assigned Numbers" [RFC1700]. This value is encrypted together with the payload, and may only be examined after decryption.

2.1. Stream Offset
-----------
Current stream ciphers were not designed for integration in IP. Therefore, provisions for detecting and handling out of order delivery,
duplicate or missing packets must be taken additionally. The Stream Offset field is used at the receiving end to prepare the ciphers internal state such that decrypting is possible even in case of abovementioned events.

To make this possible, the receiver MUST be able to accept packets with Stream Offset being a LIMITED number of bytes ahead or before the previously received Stream Offset.

The sender MUST count all bytes encrypted before the first byte in this packet is encrypted and send this value as this packet’s Stream Offset. The counter MUST be initialized with zero when the encryption key is used the first time. It MUST be reset to zero if and only if the encrypting key changes. It MUST NOT be reset at any other time. Especially, it MUST NOT be allowed to wrap around to zero, the encryption key MUST be changed instead. It is most important (see security considerations in section 5), that no encrypting key is used with the same Stream Offset twice. Thus, after the encrypting key has been changed, the probability for its reuse MUST be as small as practically feasible. Ideally, an old encryption key SHOULD NOT be reused ever.

The sender MAY encrypt a LIMITED number of dummy bytes between packets, although this is discouraged, because it would use more cache space at the receiver.

2.2. Replay Protection Using the Stream Offset
-----------------------------------------------

Since the sender creates Stream Offsets so that for any two packets encrypted with the same key their ranges \([\text{Stream Offset} \ldots (\text{Stream Offset} + \text{Encrypted Bytes in Packet} - 1)]\) are free of intersections, the receiver may optionally check this range independence to protect against replay attacks. The receiving end may independently decide whether replay protection is to be performed or not, it is not mandatory that the sender be involved in this decision.

To achieve Replay Protection, the receiver has to remember the ‘‘holes’’ in the Stream where incoming packets may still be accepted. For each currently employed key, a LIMITED list of valid Stream Offset ranges has to be stored. They cover the range starting from the end of the ‘highest’ received packet up to the maximum offset, and ranges where packets have not yet been delivered (e.g. were lost in transit or will be delivered out of order). In section 4 we will give pseudo code that implements replay protection for the receiving end, at the same time allowing for reception of out of order packets.

2.3. Authentication Issues
---------------------------

Authentication is not included into the ‘esp-stream’ transform, mainly
because of the following points:

a) The main reason to employ stream ciphers such as RC4 or SEAL is the remarkable speed of these algorithms. MD5 software speeds are adequate for commonly deployed LAN and WAN links, but reportedly are too slow for newer link technologies [RFC1810]. If faster implementations emerge, their use is strongly recommended, as encryption without authentication offers only a very limited amount of security.

b) Authentication may be chosen as an orthogonal transform in addition to this transform, so users have the ability to choose the authentication that best fits necessities in speed and security at runtime.

3. Encryption

------------

Procedural description of the encryption process:

Encrypt packet:
- Prepare sufficient space before and after the payload;
- Append a Payload Type byte describing the type of encapsulated data;
- Store the SPI in the SPI field;
- Get the current internal state associated with this security association;
- Store the number of data bytes already encrypted with the current key into the Stream Offset field;
- Encrypt the payload and the Payload Type byte, updating the internal state;
- Generate outer headers such as AH or IP;
- Send packet;

4. Decryption

------------

Pseudo code to allow efficient handling of out of order packet reception and packet loss including replay protection is given below. To remember the range of Stream Offsets already used by the received packets, a set of 3-tuples for each packet key is stored. It contains start of used range, end of used range (=start+length, effectively being start of next expected packet), and cipher internal state associated with the Stream Offset of the packet that would immediately follow.

If no protection against replay attacks is required, the following pseudo code may be simplified. We strongly recommend using replay protection, as the additional cost is low and the benefits are large.

FORWARD_SEEK_LIMIT and STATE_CACHE_ENTRIES are defined in section 6.1, DATA_SIZE includes the actual Payload size and the one byte Payload Type field.

Key initialization (input: KEY):
- Set INTERNAL_STATE to the result of prepare_key(KEY);
Store tuple (0, 0, INTERNAL_STATE);
Return;

Decrypt packet (input: ESP_HEADER, ESP_DATA, DATA_LENGTH):
  Process all outer headers;
  Set START to Stream Offset found in ESP_HEADER;
  Set END to START + DATA_LENGTH;
  If intersection between [START..END] and any stored range {
    Drop packet and return; // Replay, garbage, or duplicate packet
  }
  Set PREDECESSOR to tuple with biggest end <= START;
  If no PREDECESSOR was found {
    Drop packet and return; // Packet too old
  }
  If START minus PREDECESSOR.END is greater than FORWARD_SEEK_LIMIT {
    Drop packet and return; // Too far out in ‘‘future’’ key stream
  }
  If START equals PREDECESSOR.END {
    Set WORK to PREDECESSOR;
  } else {
    // Seek forward to a new IV
    Duplicate of PREDECESSOR.IV, call it TEMP_IV;
    Run rc4 on PREDECESSOR.END-START dummy bytes
    with TEMP_IV, updating it;
    Store new tuple WORK = (START, START, TEMP_IV);
  }
  // We have a tuple just preceeding ours,
  // extend it by our newly received range
decrypt with WORK.IV, updating the IV;
  Set WORK.END to END;
  If there is a tuple NEIGHBOUR whose start == END {
    // join the two ranges into one
    Set NEIGHBOUR.START to WORK.START;
    Delete WORK;
  }
  If more than STATE_CACHE_ENTRIES tuples are stored {
    // delete the first (oldest) hole.
    Set START of the first remaining tuple to 0;
  }
  Get Payload Type;
  If Payload Type is valid {
    Process packet according to payload type and return;
  } else {
    Drop the packet and return;
  }

It is suggested that a new tuple (with extended range) is only stored if
the incoming packet is correctly authenticated. Alternatively, if no
authentication is required, it may also be stored if the packet decrypts
correctly (i.e. passes some integrity tests, such as a valid TCP or UDP
checksum in inner headers). This is to prevent denial of service attacks
involving excessive seeking in the key stream. In the case a packet
fails the verification the packet MUST be dropped.

5. Security Considerations
---------------------------
One goal of this document is to allow different implementors of the
stream cipher RC4 to interoperate securely. Some of the security con-
siderations discussed here hold true for other ciphers.

Do NOT re-use key material. Do not use previous keys to generate new
keys, but use a reliable and well-initialized random number generator to
generate new keys. See [RFC1750] for additional information. If you do
re-use key material, the security of a stream cipher is instantly lost.

Remove all information belonging to old keys whenever they become
invalid, including information held in caches, especially the ‘‘hole’’
cache. The key management protocol in the upper layer has to decide when
a key becomes invalid.

As pointed out in [Roos95], there are classes of weak keys (especially
those whose first two bytes are two’s complements of each other), which
SHOULD be avoided in RC4.

Since the first few bytes generated after the initial setup of the RC4
are those easiest cryptanalized and broken (see [Roos95]), we recommend
that the sender SHOULD encrypt a LIMITED number of dummy bytes (updating
the internal state and the Stream Offset accordingly) before starting to
send packets. Receivers must be prepared to accept a first packet with a
Stream Offset that is equal to a LIMITED number of bytes above zero.

We strongly recommend using authentication if your security requirements
are high and you want to protect against active attacks, such as those
described in [Bell96]. This paper also gives a good overview over the
risks and open security problems that need to be taken into considera-
tion.

6. Implementation Details
--------------------------
6.1. Summary of Limits
-----------------------
In the following table we show up to three values or ranges for every
variable: the minimum supported number (what each implementation MUST
support), some recommended values (what implementations SHOULD use), and
a maximum that MUST NOT be exceeded. The column ‘‘Who’’ indicates
whether the sender (S) or the recipient (R) or both need to support these limits. Values are given in bytes, except where otherwise noted.

<table>
<thead>
<tr>
<th>Name</th>
<th>Who</th>
<th>Minimum (MUST)</th>
<th>Recommended (SHOULD)</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key length (bits)</td>
<td>SR</td>
<td>128</td>
<td>128 .. 2048</td>
<td>2048</td>
</tr>
<tr>
<td>Initial forward seek</td>
<td>S</td>
<td>0</td>
<td>16 .. 1024</td>
<td>64K</td>
</tr>
<tr>
<td>Initial forward seek</td>
<td>R</td>
<td>64K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forward seek per packet</td>
<td>R</td>
<td>32K .. 200K</td>
<td></td>
<td>512K</td>
</tr>
<tr>
<td>Backward history</td>
<td>R</td>
<td>16K .. 100K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>State cache entries per key</td>
<td>R</td>
<td>4 .. 16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes
-----

Key length
indicates the number of bits that an implementation must support, although the use of shorter keys may be required in certain circumstances.

Initial forward seek
indicates the number of bytes the sender should seek forward before encrypting the first packet. The receiver must accept a packet encrypted with a new key with Stream Offsets going up to the value of ‘‘initial forward seek’’. Be aware that the first packet encrypted with the new key can get lost, so that senders cannot exploit the full limit on ‘‘initial forward seek’’ that receivers have.

Forward seek per packet
indicates the amount of bytes that the receiver allows the Stream Offset of two succeeding packets to be apart, thus restricting the amount of forward or backward seeking that may be performed. Otherwise, a denial of service attack could be easily mounted by forcing the receiver to do exhaustive seeking in the cipher key stream.

Backward history
indicates the amount of past bytes for which the receiver should be able to reconstruct the internal state of any ‘‘hole’’ (needed for out of order delivery).

State cache entries per key
is the recommended number of previous internal states a receiver should keep. Under normal circumstances this is enough to provide good coverage of swapped and delayed packets.
6.2. Technical Notes
---------------------
To prevent getting a blocked channel whenever packets are received out of order, receivers SHOULD maintain multiple internal states for the stream cipher, and handle these conditions gracefully.

It is also suggested, that the stream cipher is rekeyed frequently (after short amounts of elapsed time) so that connections can easily recover after excessive blocks of data have been lost. Otherwise the limit on forward seeking at the receiving end might block the connection.

Seeking forward by n bytes in the cipher keystream is identical to encrypting that many dummy bytes. Forward seeks may be optimized by creating a dedicated procedure which only implements the relevant parts of the algorithm. Sophisticated solutions may want to add the capability for backward seeking in algorithms where this is possible (such as RC4), and can then use the nearest stored state to seek to the needed position. When algorithms (such as SEAL) are used, where the current position in the stream can be set, use of this functionality is strongly encouraged.

7. Acknowledgements
---------------------
The authors would like to thank Ran Atkinson <rja@cisco.com> for his helpful comments and the authors of RFCs 1826, 1828, and 1851, from which many ideas were used.

Special thanks go to Christian Schneider <cschneid@amiga.icu.net.ch> and Michael Hauber <hauber@amiga.icu.net.ch> for incorporating stream cipher support into ENskip and the extensive evaluation of replay protection.

The SKIP team at SUN-ICG (Ashar Aziz, Tom Markson, Martin Patterson and Josef Reveane) originally suggested the use of RC4 in the early SKIP drafts, and proved to be an extremely valuable partner in interoperability tests during public forums.

We express our gratitude to Ron Rivest for pointing out legal aspects relating to our original ‘alleged RC4’ draft, and, nevertheless, for devising such a simple and elegant cipher such as RC4.

8. References
--------------


Roos95 A. Roos <andrewr@vironix.co.za>, "Weak Keys in RC4", Posted to sci.crypt on Sep 22, 1995, archived in e.g. http://www.tik.ee.ethz.ch/~mwa/RC4/WeakKeys.txt


9. Author addresses
---------------------
Questions and comments about this memo can be directed to:

Germano Caronni
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH)
ETH Zentrum
CH-8092 Zurich

caronni@tik.ee.ethz.ch, OR gec@acm.org

Marcel Waldvogel
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH)
ETH Zentrum
CH-8092 Zurich