Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 5, 2013

J. Snell
August 4, 2012

HTTP/2.0 Discussion: SPDY In-Session Key Negotiation
draft-snell-httpbis-keynego-00

Abstract

This memo describes a proposed modification to SPDY that introduces the concepts of In-Session Key Negotiation and Secure Framing.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

In-Session Key Negotiation allows endpoints to dynamically negotiate cryptographic keys after a SPDY Session has already been established through the exchange of one or more KEY_NEGO control frames.

There are a number of benefits to such a mechanism:

1. The ability to negotiate multiple keys over a single TCP/IP connection.
2. The ability to renegotiate keys on the fly without tearing down and reestablishing the TCP/IP connection.
3. Key Negotiation is intermediary friendly while remaining secure. Both Hop-by-Hop and End-to-End negotiation schemes would be possible.
4. Support for multiple key negotiation mechanisms, including pre-shared key.
5. Support for server-initiated key negotiation .. allowing responses to be secured on-demand by servers even if the client did not initiate the secure request. This allows servers to enforce secure communication with the client.
6. The ability to target specific key negotiations at individual hosts.
7. The possibility of using negotiated keys as an alternative to basic and digest authentication.

TODO: More coverage on the needs, benefits

2. In-Session Key Negotiation

The KEY_NEGO control frame is used to negotiate cryptographic keys for use by either endpoint within an established SPDY Session.

The KEY_NEGO Frame

```
+----------------------------------+
|1| version   |      KEY_NEGO      |
+----------------------------------+
| Flags (8)  |   LENGTH (24)       |
+----------------------------------|
|X|          KEY_ID (31)           |
+----------------------------------+
|X|  Associated-To-Stream-ID (31) |
+----------------------------------+
| ALG_ID(16) | SEQ(8)    |         |
+------------------------+         |
|         (HEADERS BLOCK)          |
|                                  |
```

Flags: Flags related to this frame. Valid flags are:
0x01 = FLAG_EXPECTS_RESPONSE - Indicates that the sender is expecting to receive a KEY_NEGO frame in response to this one.
0x02 = FLAG_DONE - Indicates that this is the last KEY_NEGO frame the sender will send for this key negotiation sequence.
0x04 = FLAG_WAIT - Indicates that the sender will be sending additional KEY_NEGO frames and that the recipient should wait for those before responding.
0x08 = FLAG_ERROR - Indicates that an error has occurred within the key negotiation sequence and that the headers contains the details of the error.
0x10 = FLAG_VOID - Indicates that the sender wishes for a previously negotiated key to be voided, making it unavailable for further use within the same SPDY Session.

Length: The length is the number of bytes which follow the length field in the frame. For KEY_NEGO frames, this is 7 bytes plus the length of Headers block.

KEY_ID: The 31-bit identifier for the key being negotiated. KEY_NEGO frames initiated by the client MUST have an odd-numbered ID. KEY_NEGO frames initiated by the server MUST have an even-numbered ID.

Associated-To-Stream-ID: The 31-bit identifier for a Stream for which this key is to be associated. If this key is independent of all other streams, it should be 0.

If a key is associated with a given stream, the key is destroyed when the stream is concluded.

ALG_ID: The 16-bit identifier of the key negotiation algorithm being performed.

SEQ: An 8-bit unsigned integer incremented for each KEY_NEGO frame exchanged for a given KEY_ID.

HEADERS BLOCK: The block of headers carried as part of the KEY_NEGO frame.

Within any single SPDY session, multiple KEY_NEGO exchanges may occur. However, once the range of possible KEY_ID's has been exhausted, no further negotiation is possible within that session.

3. Secure Framing

Obviously, negotiating a key is pointless if it cannot be

subsequently used to secure communications. For this, we can either modify the existing SPDY frames defined in [draft-mbelshe-httpbis-spdy-00] or introduce additional extension Control Frames. Currently, this memo adopts the latter approach.

Three new Control Frames would be introduced:

- o SYN_SEC_STREAM
- o SYN_SEC_REPLY
- o INTEGRITY

The SYN_SEC_STREAM and SYN_SEC_REPLY control frames are generally identical to the existing SYN_STREAM and SYN_REPLY frames, but include an additional 31-bit KEY_ID field that identifies the negotiated key used to encrypt the contents of both the block of headers (within the SYN_* frame as well as subsequent HEADERS frames) and all data frames within the stream.

SYN_SEC_STREAM Control Frame:

```
+----------------------------------+
|1|    version    |  SYN_SEC_STREAM |
+----------------------------------+
|  Flags (8)  |  Length (24 bits)   |
+----------------------------------+
|X|           Stream-ID (31bits)    |
+----------------------------------+
|X| Associated-To-Stream-ID (31bits) |
+----------------------------------+
| Pri|Unused | Slot |X| KEY_ID (31)  |
+----------------------------------|
|           (Headers Block)         |
|                ...                |
```

SYN_SEC_REPLY:

```
+----------------------------------+
|1|    version    |   SYN_SEC_REPLY |
+----------------------------------+
|  Flags (8)  |  Length (24 bits)   |
+----------------------------------+
|X|           Stream-ID (31bits)    |
+----------------------------------+
|X|             KEY-ID (31)         |
+----------------------------------+
|           (Headers Block)         |
|                ...                |
```

Additional, a new Stream Integrity Control frame is proposed that

allows a sender to periodically insert a checksum into the stream. The checksum is calculated over the bytes of all HEADERS and Data frames sent since (and including) the initial SYN_* control frame or the previously sent INTEGRITY frame. If a key is used to generate the digest, the KEY_ID field can be used to reference the key. If the SYN_SEC_STREAM or SYN_SEC_REPLY contained a KEY_ID, then the digest is encrypted using the identified key..

INTEGRITY Frame:

```
+----------------------------------+
|0| version |      INTEGRITY       |
+----------------------------------+
|X|      Stream-ID (31bits)        |
+----------------------------------+
|X|         KEY-ID (31bits)        |
+----------------------------------+
| ALG_ID (8) | SEQ(8) |Length (24) |
+----------------------------------+
|             Digest               |
+----------------------------------+
```

If the recipient receives an INTEGRITY frame that does not validate, it can choose to terminate the stream with a RST_STREAM.

4. Example: Pre-shared Secret Key

Consider a scenario where user, Tom, is accessing a service on host "example.org". As part of the out of band registration process, a shared secret key is generated and shared by Tom and the hosted service. This key is tied to Tom's user account name: "tom".

In this example, only a single KEY_NEGO frame needs to be exchanged, sent by Tom to the Server to identify the name of the pre-shared key.

```
     Tom                    Server
      |                       |
      |=====================>|
      | 1) SYN                |
      |<=====================|
      | 2) SYN_ACK            |
      |=====================>|
      | 3) ACK                |
      |                       |
      |=====================>|
      | 4) KEY_NEGO           |
      |  ID=1                 |
      |  ALG_ID=1 (PSK)       |
      |  FLAGS=0x02           |
      |  SEQ=1                |
      |  :host=example.org    |
      |  :key=tom             |
      |                       |
      |=====================>|
      | 5) SYN_SEC_STREAM     |
      |   ID=1                |
      |   KEY_ID=1            |
      |   :method=POST        |
      |   :host=example.org   |
      |                       |
      |=====================>|
      | 6) DATA               |
      |  ID=1                 |
      |   (encrypted data)    |
      |                       |
      | ...
```

The SYN_SEC_STREAM establishes a secured stream that references the established key, and all headers and data transmitted would be encrypted using the identified key.

The server MAY choose to respond with either a SYN_REPLY or SYN_SEC_REPLY.

5. Example: Diffie-Helmman Exchange

Multi-step key negotiation mechanisms, such as the popular Diffie-Hellman mechanism, can also be implemented through the exchange of multiple KEY_NEGO frames.

```
     Tom                  Server
      |                      |
      |=====================>|
      | 1) SYN               |
      |<=====================|
      | 2) SYN_ACK           |
      |=====================>|
      | 3) ACK               |
      |                      |
      |=====================>|
      | 4) KEY_NEGO          |
      |  ID=1                |
      |  ALG_ID=2 (DH)       |
      |  FLAGS=0x01          |
      |  SEQ=1               |
      |  :host=example.org   |
      |  :p={p}              |
      |  :g={g}              |
      |  :A={A}              |
      |                      |
      |<=====================|
      | 5) KEY_NEGO          |
      |  ID=1                |
      |  ALG_ID=2 (DH)       |
      |  FLAGS=0x02          |
      |  SEQ=2               |
      |  :B={B}              |
      |                      |
      |<====================>|
      |    STREAM / REPLY    |
      |   (secured w/Key 1)  |
      |                      |
```

6. Example: In-Session TLS

KEY_NEGO frames can even be orchestrated to mimic the existing TLS-Handshake protocol:

```
     Tom                  Server
      |                      |
      |=====================>|
      | 1) SYN               |
      |<=====================|
      | 2) SYN_ACK           |
      |=====================>|
      | 3) ACK               |
      |                      |
```

```
      |=====================>|
      | 4) KEY_NEGO          | // CLIENT_HELLO
      |  ID=1                |
      |  ALG_ID=3 (IS-TLS)   |
      |  FLAGS=0x01          |
      |  SEQ=1               |
      |  :host=example.org   |
      |  :gmt_unix_time={X}  |
      |  :random:...         |
      |  :session:...        |
      |  :ciphers:...        |
      |  :extensions:...     |
      |                      |
      |<=====================|
      | 5) KEY_NEGO          | // SERVER_HELLO
      |  ID=1                |
      |  ALG_ID=3            |
      |  FLAGS=0x04          |
      |  SEQ=2               |
      |  :random:...         |
      |  :session:...        |
      |  :cipher:...         |
      |  :extensions:...     |
      |  :cert:...           |
      |  ...                 | <==| Certificate
      |                      | <==| ServerKeyExchange
      |                      | <==| CertificateRequest
      |<=====================|
      | 6) KEY_NEGO          | // SERVER_FINISHED
      |  ID=1                |
      |  ALG_ID=3            |
      |  FLAGS=0x2           |
      |                      | |==> Certificate
      |                      | |==> ClientKeyExchange
      |                      | |==> CertificateVerify
      |                      | <==> Change Cipher Spec
      |=====================>|
      | 7) KEY_NEGO          | // CLIENT_FINISHED
      |  ID=1                |
      |  ALG_ID=3            |
      |  FLAGS=0x2           |
      |                      |
      |<====================>|
      |    STREAM / REPLY    |
      |   (secured w/Key 1)  |
      |                      |
```

7. Example: Server-Initiated Key Exchange

One of the more interesting cases enabled by In-Session Key Negotiation is the possibility of server-initiated protection. That is, if a client opens an insecured stream with the server, the server could choose to upgrade that stream on-the-fly by initiating a KEY_NEGO exchange and responding with a SYN_SEC_REPLY. All content returned by the server would be encrypted, even if the request was not.

```
  Tom                    Server
   |                       |
   |=====================>|
   | 1) SYN                |
   |<=====================|
   | 2) SYN_ACK            |
   |=====================>|
   | 3) ACK                |
   |                       |
   |=====================>|
   | 4) SYN_STREAM         |
   |  ID=1                 |
   |  :method=GET          |
   |  :path=/              |
   |  :host=example.org    |
   |                       |
   |<=====================|
   | 5) KEY_NEGO           |
   |  ID=2                 |
   |  ASSOC_STREAM_ID=1    |
   |  ALG_ID=1             |
   |  FLAGS=0x2            |
   |  :key="tom"           |
   |                       |
   |<=====================|
   | 6) SYN_SEC_REPLY      |
   |  ID=1                 |
   |  KEY_ID=2             |
   |  ...                  |
   |                       |
```

8. Security Considerations

TBD. TODO: Need to expand this...

Negotiated Keys should likely be tied to a same-origin policy. The same negotiated key could not be used with multiple origins...

instead, require the client to negotiate a separate key for each origin unless the specific key negotiation protocol allows multi-origin operation.

9. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Author's Address

James M Snell

Email: jasnell@gmail.com