Secure RTP Library API Documentation

David A. McGrew
Cisco Systems, Inc.

Contents

Chapter 1

Overview

This document describes the API for libsrtp, the Open Source Secure RTP library from Cisco Systems, Inc. RTP is the Real-time Transport Protocol, an IETF standard for the transport of real-time data such as telephony, audio, and video, defined by RFC1889. Secure RTP (SRTP) is an RTP profile for providing confidentiality to RTP data and authentication to the RTP header and payload. SRTP is currently an Internet Draft in the IETF Audio/Video Transport Working Group.

Libsrtp provides functions for protecting RTP sessions. RTP packets can be encrypted and authenticated (using the `srtp_protect()` function), turning them into SRTP packets. Similarly, SRTP packets can be decrypted and have their authentication verified (using the `srtp_unprotect()` function), turning them into RTP packets.

The typedef `srtp_t` points to a structure holding all of the state associated with an SRTP stream, including the keys and parameters for cipher and message authentication functions and temporary storage used in processing the data. This pointer is the first argument to all of the libsrtp functions. A particular `srtp_t` holds the information needed to protect a particular RTP and RTCP stream. Recall that an RTP session is defined by a pair of destination transport address, e.g. the IP address and UDP port numbers of the RTP and RTCP. Within a session, there can be multiple streams, each originating from a particular sender. Each sender has a distinct SRTP context with a distinct master key used to protect the RTP and RTCP streams that it is originating. The `srtp_t` is a handle to the context used to protect a particular RTP and RTCP stream.

The enum `sec_serv_t` describes the security services that SRTP can provide. It is provided to `srtp_alloc()`, when a `srtp_t` is initialized, in order to specify the services that will be applied by `srtp_protect()` or be enforced by `srtp_unprotect()`. The values and their meanings are given in the following table.

Value	Meaning
`sec_serv_none`	No security services are provided
`sec_serv_conf`	RTP payload confidentiality is provided
`sec_serv_auth`	RTP packet authentication is provided
`sec_serv_conf_and_auth`	RTP packet authentication and payload confidentiality provided

The structure `srtp_hdr_t` contains an RTP or SRTP header (both formats are identical). **This structure is assumed to be aligned on a four-byte boundary.**

An `srtp_t` is as allocated using the function `srtp_alloc()` and then initialized using the function `srtp_init_aes_prf()`. After it is initialized, it can be used to protect RTP packets using the `srtp_protect()` function, and to protect RTCP packets using the `srtp_protect_rtcp()` function. Alternatively, it can be used to unprotect SRTP or SRTCP packets using the `srtp_unprotect()` and `srtp_unprotect_srtcp()` functions. A particular `srtp_t` should only be used for protection or for unprotection, but *not* for both. Like most cryptographic protocols, Secure RTP requires that only one device use a particular key to encrypt outbound traffic. To ensure that this property is maintained, each RTP/RTCP stream **must** be protected by a distinct `srtp_t`, and each `srtp_t` **must** be used to protect a single stream (or unprotect a single stream, on the receiver's side).

Chapter 2

Secure RTP Functions

srtp_protect()

```
err_status_t srtp_protect(srtp_t ctx, srtp_hdr_t *pkt, int *pkt_octet_len)
```

The function srtp_protect(ctx, pkt, len) applies secure rtp protection to the packet pkt (which has length *len) using the srtp context ctx. It is the srtp sender-side packet processing function. The arguments are:

ctx is a pointer to the srtp_t which applies to the particular packet

pkt is a pointer to the header of the rtp packet to be secured; after the function returns, it points to the srtp packet

pkt_octet_len is a pointer to the length in octets of the complete RTP packet (header and body) before the function call, and of the complete SRTP packet after the function returns

The return values are

err_status_ok no problems occured

err_status_replay_fail the RTP sequence number was used before (e.g. is non-increasing); repeat values cannot be used by SRTP

other failure in cryptographic mechanisms

Note: this function may write data at the end of the packet. There must be writeable memory at the end of the packet to hold this data, though this memory need

not be initialized. The length of this data is constant for a given `srtp_t` and can be found by using the function `srtp_get_trailer_length()`.

srtp_unprotect()

```
err_status_t srtp_unprotect(srtp_t ctx, srtp_hdr_t *pkt, int *pkt_octet_len)
```

The function srtp_unprotect(ctx, pkt, len) applies secure rtp protection to the RTP packet pointed to by pkt (which has length *len), using the srtp contet pointed to by ctx. This is the secure rtp receiver-side packet processing function. The arguments are:

ctx is the srtp_t context to apply to the particular packet

pkt is a pointer to the srtp packet (before the call). after the function returns, it points to the rtp packet if err_status_ok was returned; otherwise, the value of the data to which it points is undefined.

pkt_octet_len is a pointer to the length in octets of the complete srtp packet (header and body) before the function call, and of the complete rtp packet after the call, if err_status_ok was returned. otherwise, the value of the data to which it points is undefined.

The return values are

err_status_ok the rtp packet is valid

err_status_auth_fail the srtp packet failed message authentication

err_status_replay_fail the srtp packet is a replay (this packet has already been processed

other failure in cryptographic mechanisms

If err_status_ok is returned, then pkt points to the RTP packet and pkt_octet_len is the number of octets in that packet; otherwise, no assumptions should be made about either data elements.

srtp_get_trailer_length()

```
int srtp_get_trailer_length(const srtp_t a)
```

The function srtp_get_trailer_length(&a) returns the number of octets that will be added to an RTP packet by the SRTP processing. This value is constant for a given srtp_t (i.e. between initializations).

srtp_alloc()

```
err_status_t srtp_alloc(srtp_t *ctx,
                        cipher_type_t *ctype,
                        int cipher_key_len,
                        auth_type_t *atype,
                        int auth_key_len,
                        int auth_tag_len,
                        sec_serv_t sec_serv)
```

The function srtp_alloc() allocates a secure rtp context given a pointer to an srtp_t. The other arguments are parameter values providing details on the SRTP options provided by the context. The arguments are:

ctx the address of the srtp_t which will point to the structure that is allocated

ctype the type of cipher to be used (see the description of cipher_type_t below)

cipher_key_len the cipher's key length in octets

atype the type of authentication function to be used (see the description of the auth_type_t below)

auth_key_len the authentication key length in octets

auth_tag_len the authentication tag length in octets

sec_serv the security services flag (defined above)

The return values are:

err_status_ok no problems

err_status_alloc_fail a memory allocation failure occured

The cipher and authentication function parameters are described in Chapter 3. **For now, please look in the file `test/srtp-driver.c` for an example usage of these parameters.**

After an srtp_t is allocated, it must be initialized by calling the srtp_init_aes_128_prf(...) function (see below). An alternative but **unrecommended** method is to initialize the cipher_t and auth_t using the cipher and auth APIs directory.

srtp_dealloc()

```
err_status_t srtp_dealloc(srtp_t ctx)
```

srtp_dealloc(ctx) deallocates storage for an srtp_t. this function should be called exactly once to deallocate the storage allocated by the function call srtp_alloc(&ctx).

The return values are:

err_status_ok no problems

err_status_dealloc_fail a memory deallocation failure occured

srtp_init_aes_128_prf()

```
err_status_t srtp_init_aes_128_prf(srtp_t srtp, const octet_t key[16], const octet_t salt[14]);
```

The function srtp_init_aes_128_prf(ctx, key, salt) initializes an srtp_t with a given master key and master salt.

The PRF used for key derivation here is that defined in the secure rtp specification, though in theory other PRFs can be used. The cipher is hardwired to AES-128 Counter Mode with 112 bits of salt.

srtp_protect_srtcp()

```
err_status_t srtp_protect_rtcp(srtp_t ctx, srtcp_hdr_t *pkt, int *pkt_octet_len)
```

This function has not yet been implemented. It will work like `srtp_protect()`.

srtp_unprotect_srtcp()

```
err_status_t srtp_unprotect_rtcp(srtp_t ctx, srtcp_hdr_t *pkt, int *pkt_octet_len)
```

This function has not yet been implemented. It will work like `srtp_unprotect()`.

srtp_get_srtcp_trailer_length()

This function is not yet implemented. It will work like `srtp_get_trailer_length()`.

Chapter 3

Cryptographic Functions

Libsrtp uses a self-contained cryptographic kernel containing ciphers and message authentication functions. A future version of libsrtp will allow the use of cryptographic mechanisms other than the default mechanisms of AES_128 and TMMHv2_4.

A `cipher_type_t` is a cipher algorithm type. A `cipher_t` is a particular cipher, containing a particular key and parameters.