





## Preface

The original implementation and documentation of libSRTP was written by David McGrew of Cisco Systems, Inc. in order to promote the use, understanding, and interoperability of Secure RTP. Randell Jesup contributed a working





|          |   |           |
|----------|---|-----------|
| <b>4</b> | <b>libSRTP Data Structure Index</b>   | <b>13</b> |
| 4.1      | libSRTP Data Structures . . . . .   | 13        |
| <b>5</b> | <b>libSRTP Module Documentation</b>   | <b>15</b> |
| 5.1      | Secure RTP . . . . .  | 15        |
| 5.2      | Secure RTCP . . . . .   | 27        |
| 5.3      | SRTP events and callbacks . . . . .   | 29        |
| 5.4      | CryptographI0(.)-558Td(27)]TJET1rg001RG1001-720(.)-500(.)-500(.)-500(.)-500(.)-500(.)-500(.)-500(.)-5 |           |

libSRTP

CONTENTS





## Chapter 1

# Introduction to libSRTP

This document describes 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 an IETF Proposed Standard, and is defined in RFC 3711, and was developed in the IETF Audio/Video Transport (AVT) Working Group. This library supports all of the mandatory features of SRTP, but not all of the optional features. See the [Supported Features](#) section for more detailed information.

This document is organized as follows. The first chapter provides background material on SRTP and overview of libSRTP. The following chapters provide a detailed reference to the libSRTP API and related functions. The reference

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
-



## 1.4 Applications

Several test drivers and a simple and portable srtp application are included in the

```
[sh1]$ test/rtpw -s -k $k -ea 0.0.0.0 9999
Security services: confidentiality message authentication
set master key/salt to C1EEC3717DA76195BB878578790AF71C/4EE9F859E197A414A78D5ABC7451
setting SSRC to 2078917053
sending word: A
sending word: a
sending word: aa
sending word: aal
sending word: aalii
sending word: aam
sending word: Aani
sending word: aardvark
...

[sh2] set k=c1eec3717da76195bb878578790af71c4ee9f859e197a414a78d5abc7451
[sh2]$ test/rtpw -r -k $k -ea 0.0.0.0 9999
security services: confidentiality message authentication
set master key/salt to C1EEC3717DA76195BB878578790AF71C/4EE9F859E197A414A78D5ABC7451
19 octets received from SSRC 2078917053 word: A
19 octets received from SSRC 2078917053 word: a
20 octets received from SSRC 2078917053 word: aa
21 octets received from SSRC 2078917053 word: aal
...
```

be distinct. This requirement can be enforced by using the convention that each SRTP and SRTCP key is used for encryption by only a single sender. In other words, the key is shared only across streams that originate from a particular device (of course, other SRTP participants will need to use the key for decryption). libSRTP supports this enforcement by detecting the case in which a key is used for both inbound and outbound data.

## 1.6 libSRTP Overview

libSRTP provides functions for protecting RTP and RTCP. RTP packets can be encrypted and authenticated (using the [srtplib\\_protect\(\)](#)

## 1.7 Example Code

This section provides a simple example of how to use libSRTP. The example code lacks error checking, but is func-









libSRTP

## Chapter 3







## Chapter 5

# libSRTP Module Documentation

### 5.1 Secure RTP

libSRTP provides functions for protecting RTP and RTCP. See Section









stream, a pointer to the SRTP master key for that stream, the SSRC describing that stream, or a flag indicating a 'wild-card' SSRC value, and a 'next' field that holds a pointer to the next element in the list of policy elements, or NULL if it is the last element.

The wildcard value `SSRC_ANY_INBOUND` matches any SSRC from an inbound stream that for which there is no explicit SSRC entry in another policy element. Similarly, the value `SSRC_ANY_OUTBOUND` will matches any SSRC from an outbound stream that does not appear in another policy element. Note that wildcard SSRCS &b cannot



### 5.1.5.2 void crypto\_policy\_set\_aes\_cm\_128\_hmac\_sha1\_32 ([crypto\\_policy\\_t](#) *p*)

**Parameters:**

*p* is a pointer to the policy structure to be set to the default policy.

The function call `crypto_policy_set_aes_cm_128_hmac_sha1_32(&p)` sets the [crypto\\_policy\\_t](#) at location `p` to use policy `AES_CM_128_HMAC_SHA1_32` as defined in `draft-ietf-mmusic-sdescriptions-12.txt`. This policy uses AES-128 Counter Mode encryption and HMAC-SHA1 authentication, with an authentication tag that is only 32 bits long.

**Returns:**  
void.

**5.1.5.4** void `crypto_policy_set_null_cipher_hmac_sha1_80` ([crypto\\_policy\\_t](#) *p*)

**Parameters:**  
*p* is a pointer to the policy structure to be set to the default policy.

The function call `crypto_policy_set_null_cipher_hmac_sha1_80(&p)` sets the [crypto\\_policy\\_t](#) at location *p* to use



**Returns:**







## 5.2 Secure RTCP

Secure RTCP functions are used to protect RTCP traffic.

### Functions





### 5.3.1 Detailed Description

libSRTP allows a user to provide a callback function to handle events that need to be dealt with outside of the data plane (see the enum `srtp_event_t` for a description of these events). Dealing with these events is not a strict necessity; they are not security-critical, but the application may suffer if they are not handled. The function `srtp_set_event_handler()` is used to provide the callback function.

A default event handler that merely reports on the events as they happen is included. It is also possible to set the event handler function to `NULL`, in which case all events will just be silently ignored.

### 5.3.2 Typedef Documentation

#### 5.3.2.1 typedef struct



## 5.4 Cryptographic Algorithms

### Modules

-









## 5.6 Authentication Function Types







## 5.8 Cryptographic Kernel

### Modules

- [Ciphers](#)

*A generic cipher type enables cipher agility, that is, the ability to write code that runs with multiple cipher types. Ciphers can be used through the crypto kernel, or can be accessed directly, if need be.*

### 5.8.1 Detailed Description

All of the cryptographic functions are contained in a kernel.













libSRTP 1.3 /Users/mcgrew/Code/sourceforge/srtp/crypto/include/ Directory Reference

libSRTP

int

## Chapter 7

# libSRTP Data Structure Documentation

### 7.1 `crypto_policy_t` Struct Reference

`crypto_policy_t` describes a particular crypto policy that can be applied to an SRTP stream.

#### Data Fields

- [cipher\\_type\\_id\\_t](#) `cipher_type`
- [int](#) `cipher_key_len`
- [auth\\_type\\_id\\_t](#) `auth_type`
- [int](#) `auth_key_len`
- [int](#)

## 7.1.2 Field Documentation

### 7.1.2.1 `int crypto_policy_t::auth_key_len`

The length of the authentication function key in octets.

### 7.1.2.2 `int crypto_policy_t::auth_tag_len`

The length of the authentication tag in octets.

### 7.1.2.3 `auth_type_id_t crypto_policy_t::auth_type`

An integer representing the authentication function.

### 7.1.2.4 `int crypto_policy_t::cipher_key_len`

The length of the cipher key in octets.

### 7.1.2.5 `cipher_type_id_t crypto_policy_t::cipher_type`

An integer representing the type of cipher.

### 7.1.2.6 `sec_serv_t crypto_policy_t::sec_serv`

The flag indicating the security services to be applied.

The documentation for this struct was generated from the following file:

- `srtp.h`



## 7.2 srtp\_event\_data\_t Struct Reference

srtp\_event\_data\_t is the structure passed as a callback to the event handler function

### Data Fields

- [srtp\\_t session](#)
- [srtp\\_stream\\_t stream](#)
-







# Index

/Users/mcgreww/Code/sourceforge/srtp/crypto/ Directory  
Reference,



- event
  - srtp\_event\_data\_t, [49](#)
- event\_key\_hard\_limit
  - SRTPEvents, [31](#)
- event\_key\_soft\_limit
  - SRTPEvents, [31](#)
- event\_packet\_index\_limit
  - SRTPEvents, [31](#)
- event\_ssrc\_collision
  - SRTPEvents, [31](#)
- HMAC\_SHA1
  - Authentication, [37](#)
- key
  - srtp\_policy\_t, [50](#)
- next
  - srtp\_policy\_t, [50](#)
- NULL\_AUTH
  - Authentication, [37](#)
- NULL\_CIPHER
  - Ciphers, [34](#)
- rtcp
  - srtp\_policy\_t, [51](#)
- rtp
  - srtp\_policy\_t, [51](#)
- SEAL
  - Ciphers, [34](#)
- sec\_serv
  - crypto\_policy\_t, [48](#)
- sec\_serv\_auth
  - SRTPEvents, [20](#)
- sec\_serv\_conf
  - SRTPEvents, [20](#)
- sec\_serv\_conf\_and\_auth
  - SRTPEvents, [20](#)
- sec\_serv\_none
  - SRTPEvents, [20](#)
- sec\_serv\_t
  - SRTPEvents, [19](#)
- Secure RTCP, [27](#)
- Secure RTP, [15](#)
- session
  - srtp\_event\_data\_t, [49](#)
- SRTCP
  - srtp\_protect\_rtcp, [27](#)
  - srtp\_unprotect\_rtcp, [28](#)
- SRTPEvents
  - crypto\_get\_random, [20](#)
  - crypto\_policy\_set\_aes\_cm\_128\_hmac\_sha1\_32, [20](#)
  - crypto\_policy\_set\_aes\_cm\_128\_hmac\_sha1\_80, [18](#)
  - crypto\_policy\_set\_aes\_cm\_128\_null\_auth, [21](#)
  - crypto\_policy\_set\_null\_ciphers\_cm\_128\_hmac\_sha1\_80,

