NAME

SCT_new, SCT_new_from_base64, SCT_free, SCT_LIST_free, SCT_get_version, SCT_set_version, SCT_get_log_entry_type, SCT_set_log_entry_type, SCT_get0_log_id, SCT_set0_log_id, SCT_set1_log_id, SCT_get_timestamp, SCT_set_timestamp, SCT_get_signature_nid, SCT_set_signature_nid, SCT_get0_signature, SCT_set0_signature, SCT_set1_signature, SCT_get0_extensions, SCT_set0_extensions, SCT_set1_extensions, SCT_get_source, SCT_set_source - A Certificate Transparency Signed Certificate Timestamp

SYNOPSIS

```
#include <openssl/ct.h>

typedef enum {
    CT_LOG_ENTRY_TYPE_NOT_SET = -1,
    CT_LOG_ENTRY_TYPE_X509 = 0,
    CT_LOG_ENTRY_TYPE_PRECERT = 1
} ct_log_entry_type_t;

typedef enum {
    SCT_VERSION_NOT_SET = -1,
    SCT_VERSION_V1 = 0
} sct_version_t;

typedef enum {
    SCT_SOURCE_UNKNOWN,
    SCT_SOURCE_TLS_EXTENSION,
    SCT_SOURCE_X509V3_EXTENSION,
    SCT_SOURCE_OCSP_STAPLED_RESPONSE
} sct_source_t;

SCT *SCT_new(void);
SCT *SCT_new_from_base64(unsigned char version,
                         const char *logid_base64,
                         ct_log_entry_type_t entry_type,
                         uint64_t timestamp,
                         const char *extensions_base64,
                         const char *signature_base64);

void SCT_free(SCT *sct);
void SCT_LIST_free(STACK_OF(SCT) *a);
```

```
sct_version_t SCT_get_version(const SCT *sct);
int SCT_set_version(SCT *sct, sct_version_t version);

ct_log_entry_type_t SCT_get_log_entry_type(const SCT *sct);
int SCT_set_log_entry_type(SCT *sct, ct_log_entry_type_t entry_type);

size_t SCT_get0_log_id(const SCT *sct, unsigned char **log_id);
int SCT_set0_log_id(SCT *sct, unsigned char *log_id, size_t log_id_len);
int SCT_set1_log_id(SCT *sct, const unsigned char *log_id, size_t log_id_len);

uint64_t SCT_get_timestamp(const SCT *sct);
void SCT_set_timestamp(SCT *sct, uint64_t timestamp);

int SCT_get_signature_nid(const SCT *sct);
int SCT_set_signature_nid(SCT *sct, int nid);

size_t SCT_get0_signature(const SCT *sct, unsigned char **sig);
void SCT_set0_signature(SCT *sct, unsigned char *sig, size_t sig_len);
int SCT_set1_signature(SCT *sct, const unsigned char *sig, size_t sig_len);

size_t SCT_get0_extensions(const SCT *sct, unsigned char **ext);
void SCT_set0_extensions(SCT *sct, unsigned char *ext, size_t ext_len);
int SCT_set1_extensions(SCT *sct, const unsigned char *ext, size_t ext_len);

sct_source_t SCT_get_source(const SCT *sct);
int SCT_set_source(SCT *sct, sct_source_t source);
```

DESCRIPTION

Signed Certificate Timestamps (SCTs) are defined by RFC 6962, Section 3.2. They constitute a promise by a Certificate Transparency (CT) log to publicly record a certificate. By cryptographically verifying that a log did indeed issue an SCT, some confidence can be gained that the certificate is publicly known.

An internal representation of an SCT can be created in one of two ways. The first option is to create a blank SCT, using **SCT_new()**, and then populate it using:

- **SCT_set_version()** to set the SCT version.

  Only SCT_VERSION_V1 is currently supported.

- **SCT_set_log_entry_type()** to set the type of certificate the SCT was issued for:

  CT_LOG_ENTRY_TYPE_X509 for a normal certificate. **CT_LOG_ENTRY_TYPE_PRECERT** for a pre-certificate.

- **SCT_set0_log_id()** or **SCT_set1_log_id()** to set the LogID of the CT log that the SCT came from.

  The former takes ownership, whereas the latter makes a copy. See RFC 6962, Section 3.2 for the definition of LogID.

- **SCT_set_timestamp()** to set the time the SCT was issued (time in milliseconds since the Unix Epoch).

- **SCT_set_signature_nid()** to set the NID of the signature.

- **SCT_set0_signature()** or **SCT_set1_signature()** to set the raw signature value.

  The former takes ownership, whereas the latter makes a copy.

- **SCT_set0_extensions()** or **SCT_set1_extensions** to provide SCT extensions.

  The former takes ownership, whereas the latter makes a copy.

Alternatively, the SCT can be pre-populated from the following data using **SCT_new_from_base64()**:

- The SCT version (only SCT_VERSION_V1 is currently supported).

- The LogID (see RFC 6962, Section 3.2), base64 encoded.

- The type of certificate the SCT was issued for: **CT_LOG_ENTRY_TYPE_X509** for a normal certificate. **CT_LOG_ENTRY_TYPE_PRECERT** for a pre-certificate.

- The time that the SCT was issued (time in milliseconds since the Unix Epoch).

- The SCT extensions, base64 encoded.

- The SCT signature, base64 encoded.

SCT_set_source() can be used to record where the SCT was found (TLS extension, X.509 certificate extension or OCSP response). This is not required for verifying the SCT.

NOTES

Some of the setters return int, instead of void. These will all return 1 on success, 0 on failure. They will not make changes on failure.

All of the setters will reset the validation status of the SCT to SCT_VALIDATION_STATUS_NOT_SET (see **SCT_validate**(3)).

SCT_set_source() will call **SCT_set_log_entry_type()** if the type of certificate the SCT was issued for can be inferred from where the SCT was found. For example, an SCT found in an X.509 extension must have been issued for a pre- certificate.

SCT_set_source() will not refuse unknown values.

RETURN VALUES

SCT_set_version() returns 1 if the specified version is supported, 0 otherwise.

SCT_set_log_entry_type() returns 1 if the specified log entry type is supported, 0 otherwise.

SCT_set0_log_id() and **SCT_set1_log_id** return 1 if the specified LogID is a valid SHA-256 hash, 0 otherwise. Additionally, **SCT_set1_log_id** returns 0 if malloc fails.

SCT_set_signature_nid returns 1 if the specified NID is supported, 0 otherwise.

SCT_set1_extensions and **SCT_set1_signature** return 1 if the supplied buffer is copied successfully, 0 otherwise (i.e. if malloc fails).

SCT_set_source returns 1 on success, 0 otherwise.

SEE ALSO

ct(7), **SCT_validate**(3), **OBJ_nid2obj**(3)

HISTORY

These functions were added in OpenSSL 1.1.0.

COPYRIGHT