Crypto Forum Research Group
Inernet Draft
Expires June, 2003

David A. McGrew
Cisco Systems, Inc.
October, 2002

The Universal Security Transform
<draft-irtf-cfrg-ust-00.txt>

Status of this Memo

This document is an Internet Draft and is in full conformance with all provisions of Section 10 of RFC-2026. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and working groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents 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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

1. Abstract

This document describes a cryptographic transform which uses an indexed keystream generator (that generates a keystream segment given an index value) and a universal hash function to provide confidentiality, message authentication, and replay protection. This transform is efficient, provably secure, and is appropriate for network security.

2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [B97].

Terms that are defined in this specification are capitalized

to distinguish them from generic terminology.

3. Universal Security Transform

The Universal Security Transform (UST) is a cryptographic transform for providing confidentiality, message authentication, and replay protection. This transform is sufficient for providing these services to network protocols, though it does not specify a protocol itself. Conceptually, it is somewhere in between a combined cipher mode of operation which provides confidentiality and authentication (such as OCB [MW]) and a completely specified security protocol (such as IPsec ESP [KA98]).

A UST input consists of an Index field and a Message field which contains the data to be protected. A UST output consists of a Ciphertext field and an Authentication Tag field. All of these fields are octet strings, and all lengths below are expressed in octets.

The Message field is divided into two parts: a Clear field, which contains data which is not altered by the transform, and an Opaque field, which is altered by the transform. Confidentiality is provided on the Opaque field through encryption, and message authentication is provided on both fields.

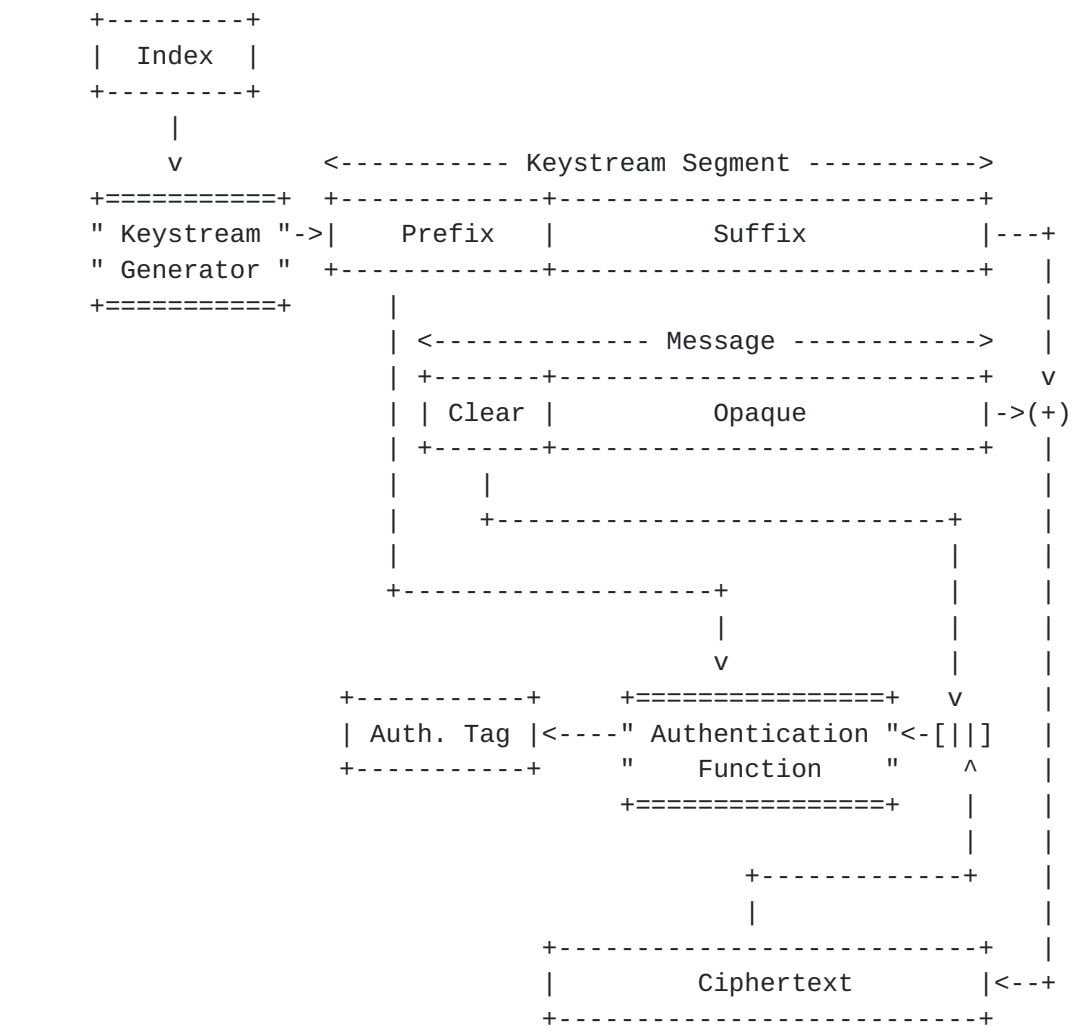
The Ciphertext contains the encrypted form of the unprotected Opaque field; the length of those fields are equal. The Authentication Tag provides message authentication of the Ciphertext and the Clear field. The Index is an unsigned integer in network byte order that acts as a nonce (that is, its value is unique for each distinct Message for each fixed key).

The UST uses a Keystream Generator (as defined in Section 5) and an Authentication Function (as defined in Section 6). The interfaces to these components, but not the components themselves, are defined in this specification. Any such components can be used.

The transform procedure is illustrated in Figure 1. The Index is fed into the Keystream Generator, which then outputs the keystream segment which corresponds to that Index. The keystream segment is conceptually divided into a Prefix, whose length is equal to that of the Authentication Tag, followed by a Suffix, whose length is that of the Opaque field. The Ciphertext is generated by bitwise exclusive-oring the Suffix into the Opaque field. The

Authentication Tag is computed by the Authentication Function, using the concatenation of the Clear field followed by the Ciphertext field as the data input and the Prefix as the masking input, as described in Section 6. (Some operations may be optional, as described in Section 3.1).

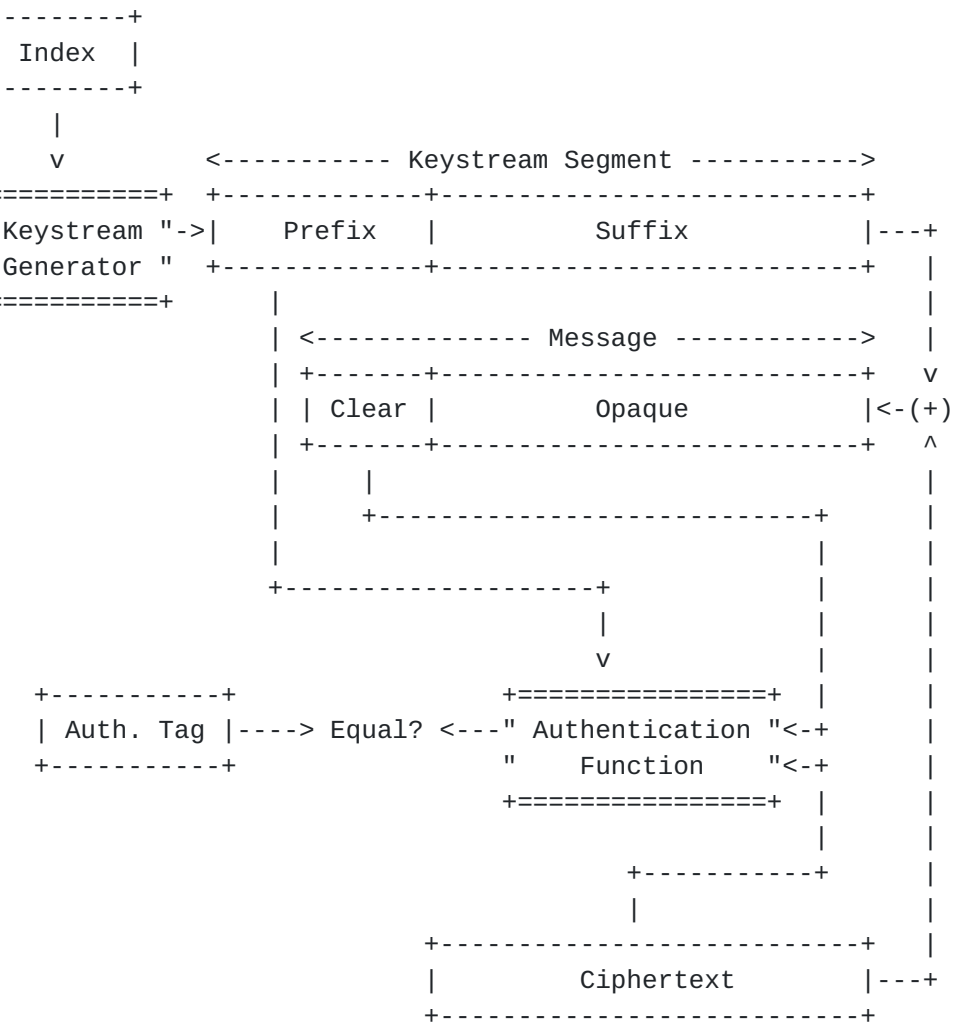
Figure 1. The UST transform. Here and below (+) denotes the bitwise exclusive-or operation and [||] denotes concatenation.



The inverse transform procedure is illustrated in Figure 2. The Index is fed into the Keystream Generator, and the Prefix is

generated. The authenticity of the message is checked by computing the value of the tag as done in the transform, and comparing the value computed with that in the Authentication Tag field of the message. If those values are equal, the Message and Authentication Tag pair is considered valid; otherwise, it is not. In the case of an authentication failure, the procedure reports an authentication error and halts. Otherwise, the Suffix is computed and is bitwise exclusive-ored into the Ciphertext, giving the Plaintext, and the procedure reports a successful authentication. (Some operations may be optional, see Section 3.1)

Figure 2. The UST inverse transform. Here "Auth. Tag" denotes the tag associated with the message; the authenticity of the message is checked by comparing that value with that of a tag computed from the other fields.



An example application programming interface is provided in Appendix A.

3.1 Options

UST can provide confidentiality, message authentication, and replay protection, or just the latter two security services. The signaling of what services are in effect for any particular use of UST are external to the transform.

When confidentiality is not provided, the Suffix MUST NOT be exored into the Plaintext or the Ciphertext.

Different messages protected with the same UST context MAY have different security services applied to them. For example, a protocol may use UST to encrypt and authenticate the data that it transports, while using the same UST context to provide only authentication to its keepalive messages.

3.2 Parameters

UST has the following parameters:

Parameter	Meaning
INDEX_LENGTH	The number of octets in an Index.
MAX_KEYSTREAM_LENGTH	The maximum number of octets in a keystream segment.
PREFIX_LENGTH	The number of octets in the keystream prefix.
TAG_LENGTH	The number of octets in an Authentication Tag.
MAX_AUTH_LENGTH	The maximum number of octets that can be input to the Authentication Function.
AUTH_KEY_LENGTH	The number of octets in an Authentication Function key.

All of these parameters MUST remain fixed for any given UST context. The parameters INDEX_LENGTH and MAX_KEYSTREAM_LENGTH are defined by the Keystream Generator. The parameters TAG_LENGTH, MAX_AUTH_LENGTH, and AUTH_KEY_LENGTH are defined by the hash function.

The length of any Plaintext protected by UST MUST NOT exceed the smaller of (MAX_KEYSTREAM_LENGTH - TAG_LENGTH) and MAX_AUTH_LEN.

The value of AUTH_KEY_LENGTH MUST be no greater than MAX_KEYSTREAM_LENGTH. The value of TAG_LENGTH MUST be no greater than AUTH_KEY_LENGTH.

3.3 Format

Unless otherwise specified, the format of the UST output is:

Here the leftmost octet denotes the first in the address range.

The octets of the Index field are the radix 256 digits of the Index value, with the leftmost octet being the most significant. The Index MAY be omitted. This option is useful when the Index can be inferred through external information, and this case is called implicit index. The other case is called explicit index. A UST implementation SHOULD provide an interface that includes both the explicit and implicit index cases. For example, a C API can provide distinct functions for each of those cases.

The ordering and encoding of the Ciphertext, Authentication Tag, and Index are unimportant for security purposes. Other specifications which specialize or adapt this one are encouraged to use formats which better suit their needs.

4. Using the UST

For each fixed UST key, each Index value MUST be distinct. This MAY be accomplished by using successive integer values (though implementers are free to use non-sequential Index values, e.g. to aid in parallelization).

The inverse transform MUST check that the value that appears in the Index has not appeared in any other inverse transform. The inverse transform MAY return a false positive (that is, report that an index has been used when in fact it has not), but MUST NOT return a false negative. The uniqueness check enforces replay protection, and false positives are allowed in order to allow implementations to reduce the amount of state which they need to maintain.

If an implicit index is used, the transform SHOULD check that the value that appears in the Index has not appeared in any previous transform. This enables a UST implementation to enforce proper security practices, rather than relying on other components of a system to meet these requirements.

4.1 UST Initialization

To initialize the context needed to use the UST transform, given a secret UST key as input, the following procedure MAY be used:

1) The Keystream Generator is initialized using the UST key.

2) The Keystream Generator is used to generate the first AUTH_KEY_LENGTH octets of the segment corresponding to the zero index. The hash function key is set to this value, and the hash function is initialized, if needed, using the hash-dependent key initialization procedure.

If this initialization method is used, then the zero index MUST NOT be used in any other invocation of the transform with that particular key, and the inverse transform MUST check that the value that appears in the Index is not zero.

5. Keystream Generators

For the purposes of UST, a Keystream Generator is an algorithm that maps a secret key and an Index to a pseudorandom keystream segment of fixed length. Each Keystream Generator MUST define the parameters INDEX_LENGTH and MAX_KEYSTREAM_LENGTH (defined in Section 3.2).

MAX_KEYSTREAM_LENGTH SHOULD be at least 65,535. This value ensures that any IP version four packet can be encrypted.

The keystream generator MUST map each possible value of the Index to a distinct value of the keystream segment, for each fixed key.

In the terms of cryptographic theory, the keystream generators used in UST are families of length-expanding pseudorandom functions. The necessary and sufficient condition on these generators is their indistinguishability from a truly random source.

6. Authentication Functions and Universal Hash Functions

An Authentication Function takes as input a data field (the Message) and a random or pseudorandom masking value (the Prefix). UST is designed for use with authentication functions which are based on universal hashing, in the Wegman-Carter paradigm [WC81]. In this method, the message is hashed using the fixed hash key, then the resulting hash value is encrypted by combining it with the prefix. The combining operation is a simple one, such as bitwise exclusive-or, and in UST it is part of the on the Authentication Function. Below we call the result of combining two values using that operation a delta.

Each Authentication Function MUST define the parameters TAG_LENGTH, PREFIX_LENGTH, AUTH_KEY_LENGTH, and MAX_AUTH_LENGTH (defined in Section 3.2).

For the purposes of UST, a universal hash function is an algorithm that maps a fixed-length secret key and a variable-length message to a fixed-length hash value, such that the delta of the hash values of distinct messages are statistically uniformly distributed. The formal mathematical requirement is that the set of functions defined by the hash with each member of the set of all possible keys is epsilon-Delta Universal [S96].

The secret key used by the universal hash is an octet string of length no greater than MAX_KEYSTREAM_LENGTH. The output of the hash function is an octet string of length TAG_LENGTH. The value PREFIX_LENGTH denotes the number of octets in the Prefix.

For use in UST, a hash function MUST be epsilon-Delta Universal (epsilon-DU) for some small value of epsilon. The value of epsilon SHOULD be close to 1/256 to the power TAG_LENGTH, so that the cryptographic strength of the tag is as large as possible. This property means that the probability that the delta of the hash of any two distinct messages will be any particular fixed value is no greater than epsilon.

The functions UHASH-16 and UHASH-32 [UMAC] meet these requirements. The functions MMH and NMH [MMH] meet all of these requirements except for the variable-length message requirement.

Note that it is technically improper to call a keyed hash function "universal". Rather, one should say that the key is an index into a universal family of hash functions. We abuse this terminology for simplicity's sake.

Authentication Functions which are not based on universal hashing MAY be used within UST. These functions can specify that the parameter PREFIX_LENGTH has a value of zero.

7. Rationale

This transform is computationally efficient, has minimal expansion, and reduces key management overhead and local state information by eliminating the need for a separate encryption key. The security properties of UST's components are well understood; a brief summary of these properties is provided in the Security Considerations Section.

The Message is divided into Clear and Opaque fields so that UST can provide message authentication but not confidentiality to some component of a message. This feature is often desirable, e.g. so that protocol headers can be protected from alteration but remain unencrypted to facilitate processing.

The benefits described above are shared with some of the recently proposed modes of operation for the Advanced Encryption Standard [AES], such as OCB, IACBC, IAPM, XCBC, and XECB modes [MODES]. However, UST has the following important advantages over those modes:

* UST can identify and reject bogus messages much faster, as it can use hash functions that can be an order of magnitude faster than AES, and authentication precedes decryption in the inverse transform. This property of UST provides it with resilience against denial of service attacks.

* UST can be operated in an authentication-only mode, whereas the other modes cannot.

* UST has minimal packet expansion.

* UST can be implemented without infringing on any patents (to the best knowledge of the authors).

Note that UST can be used with any block cipher mode that meets the requirements of Section 5, such as Counter Mode [MODES].

The benefits of universal hashing for message authentication are well known in the cryptographic literature [CW81]. The recently defined UMAC message authentication code [UMAC] uses this technique. However, these MACs do not provide confidentiality. In contrast, UST provides both security services and amortizes the per-index keystream generation cost over both services.

UST reflects implementation experience from the Secure Real-time Transport Protocol [SRTP] and the Stream Cipher ESP, a proposal for using indexed keystream generators within ESP, as well as input

from other areas.

8. Security Considerations

The security of UST follows from the indistinguishability of the keystream generator from a truly unpredictable source and the properties of the hash function. The number of unprovable assumptions which underlie the transform are thus reduced to one, the minimum number required for any cryptosystem. (Note that the converse is also true; the security of UST stands and falls on that single assumption).

Given the indistinguishability of the keystream generator, the adversary gains no knowledge about the plaintext from the ciphertext.

The probability with which an adversary can successfully forge an Authentication Tag for any given message is at most epsilon, when the hash function is epsilon-Delta Universal [S96].

The expected number N of successful forgeries is T * epsilon, where T is the number of forgery attempts, that is, the number of bogus index/ciphertext/tag values sent by the adversary to the UST receiver. The theoretical maximum value for T is (256)^INDEX_LENGTH. This value of T implies that every single message processed by the receiver is a forgery attempt. Note that if INDEX_LENGTH is greater than TAG_LENGTH, then N can be greater than one.

The resistance of an UST implementation to forgery attacks can be improved in some circumstances by limiting the number of authentication failures that will be tolerated. This limitation could be enforced by the implementation of the UST inverse transform, by maintaining a count of the total number of authentication failures and causing the inverse transform to indicate an authentication failure on all messages after the threshold has been exceeded, until a new key is derived. If no more than F failures will be tolerated by the UST inverse transform, then the expected number N of successful forgeries can be no more than F * epsilon. Of course, there is a denial of service implication in this approach which can outweigh its benefits in some scenarios.

UST permits arbitrarily small authentication tags. This is because the goal of this specification is to provide a mechanism, rather than to dictate a policy. It is expected that some applications can tolerate a one in a billion likelihood of forgery. In

particular, digital representations of analog data such as voice, audio, or video may be able to tolerate such a forgery likelihood due to the inherently imprecise nature of analog data.

9. History

This is the first draft of UST within the IRTF Crypto Forum Research Group. It is based closely on the draft draft-mcgrew-saag-ust-00.txt that was submitted to SAAG in November, 2001, which in turn was based on the draft draft-mcgrew-saag-sst-00.txt of June, 2001.

Changes from draft-mcgrew-saag-ust-00.txt to this one include a number of clarifications and corrections to the exposition.

Changes from draft-mcgrew-saag-sst-00.txt to draft-mcgrew-saag-ust-00.txt include:

* The authentication tag was previously defined as the exor of the prefix and the hash output. This definition has been changed to allow different hashes to be used within this specification, by pushing the delta combining operation into the message authentication function itself.

* The original draft did not divide the Message field into a Clear and an Opaque component.

* The UST was originally named SST. The name was changed to avoid confusion with the Shiva Smart Tunneling Protocol. The original specification was documented in draft-mcgrew-saag-sst-00.txt.

10. Acknowledgments

Thanks are due to Jesse Walker, Doug Smith, Scott Fluhrer, David Wagner, Mats Naslund, Burt Kaliski, Mark Baugher, and Raif S. Naffah for critical review and insights. Their comments significantly improved this specification.

11. Contact Information

Questions and comments on this draft SHOULD be sent to:

David A. McGrew
Cisco Systems, Inc.

mcgrew@cisco.com

and MAY be copied to the Crypto Forum Research Group at

cfrg@ietf.org

12. References

[B97] S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, RFC 2119, March 1997.

[AES] FIPS 197, The Advanced Encryption Standard, United States National Institute for Standards and Technology (NIST), http://www.nist.gov/aes/.

[CW81] M. Wegman and L. Carter, New hash functions and their use in authentication and set equality, J. of Computer and System Sciences, vol. 22, 1981.

[KA98] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[MMH] S. Halevi, and H. Krawczyk, MMH: Software Authentication in the Gbit/second rates, Fast Software Encryption Workshop, 1997. Also available online at http://www.research.ibm.com/people/s/shaih/pubs/.

[MODES] Proposed Modes of Operation, NIST web page, http://csrc.nist.gov/encryption/modes/proposedmodes/

[S96] Stinson, D. R., On the connections between universal hashing, combinatorical designs and error-correcting codes, Congressus Numerantium, 114, 1996, 7-27.

[SRTP] The Secure Real-time Transport Protocol, Blohm et. al., Work in Progress, draft-ietf-avt-srtp-02.txt.

[UMAC] Black, Halevi, Krawczyk, Krovetz, Rogaway. UMAC: Fast and Secure Message Authentication. Advances in Cryptology - CRYPTO '99. Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, 1999, pp. 216-233. Available online at http://www.cs.ucdavis.edu/~rogaway/umac/.

Appendix A. A C language API for UST.

The following C API is provided as an example of an interface to UST. This API does not provide every option or possible choice of parameters.

```
/*
 * ust.h
 *
 * C interface for the universal security transform
 *
 */

typedef unsigned char octet_t;

/*
 * a ust_ptr_t points to a structure holding the ust context
 */

typedef ust_ctx_t *ust_ptr_t;

/*
 * ust_init(...) initializes the ust context at ctx
 */

int
ust_init(ust_ptr_t ctx,              /* pointer to ust context              */
      int index_length,          /* number of octets in the index       */
      int cipher_id,             /* keystream generator identifier      */
      octet_t *cipher_key,       /* cipher key                          */
      int cipher_key_len,        /* number of octets in cipher key      */
      int auth_id,               /* auth algorithm identifier           */
      octet_t *auth_key,         /* auth key                            */
      int auth_key_len,          /* number of octets in the auth key    */
      int auth_tag_len,          /* number of octets in the auth tag    */
      int replay_window_len      /* length of replay window (0 == none) */
      );
```

```
/*
 * the ust_xfm function (ust transform)
 *
 * ctx is the ust context, which holds the cipher and hash function
 * keys and parameters, as well as anti-replay information
 *
 * idx is the packet index, a 48-bit unsigned integer which
 * should be unique for each invocation of ust_xfm for a
 * given ctx
 *
 * if auth_start != NULL, then authentication is provided to the
 * auth_len octets of data at auth_start by computing the
 * authentication tag and writing it to *tag; otherwise, the
 * authentication tag is not computed
 *
 * if enc_start != NULL, then encryption is provided to the
 * enc_len octets of data at enc_start by exoring keystream
 * into that data; otherwise, no encryption is done
 *
 * tag points to the authentication tag; after ust_xfm returns,
 * it contains the tag corresonding to the data at auth_start
 * (if auth_start != NULL) -- note that there MUST be at least
 * ust_tag_len(ctx) octets of storage at *tag!
 */

int
ust_xfm(ust_ptr_t ctx,       /* pointer to ust context          */
        xtd_seq_num_t idx,   /* index                           */
        octet_t *enc_start,  /* pointer to encryption start     */
        int enc_len,         /* number of octets to encrypt     */
        octet_t *auth_start, /* pointer to authentication start */
        int auth_len,        /* number of octets to authenticate */
        octet_t *tag         /* authentication tag              */
        );
```

```
/*
 * the ust_inv_xfm function (ust inverse transform)
 *
 * ctx is the ust context, which holds the cipher and hash function
 * keys and parameters, as well as anti-replay information
 *
 * idx is the packet index, a 48-bit unsigned integer which
 * should be unique for each invocation of ust_xfm for a
 * given ctx
 *
 * if auth_start != NULL, then authentication is expected on the
 * auth_len octets of data at auth_start by computing the
 * authentication tag and comparing it to the value at *tag;
 * otherwise, no authentication check is performed
 *
 * if enc_start != NULL, then decryption is done to the enc_len
 * octets of data at enc_start by exoring keystream into that data;
 * otherwise, no decryption is done
 *
 * tag points to the authentication tag;  if auth_start != NULL,
 * then *tag is expected to hold the authentication tag corresponding
 * to the data at *auth_start -- note that there MUST be at least
 * ust_tag_len(ctx) octets of readable data at *tag!
 */

int
ust_inv_xfm(ust_ptr_t ctx,       /* ust context                    */
         xtd_seq_num_t idx,   /* index                          */
         octet_t *enc_start,  /* pointer to encryption start    */
         int enc_len,         /* number of octets to encrypt    */
         octet_t *auth_start, /* pointer to authentication start  */
         int auth_len,        /* number of octets to authenticate */
         octet_t *tag         /* authentication tag             */
         );

/*
 * ust_tag_len(ctx) returns the length (in octets) of the
 * authentication tag for the ust context ctx.
 *
 * this function can be used to determine the storage
 * space required to hold a particular tag, if need be
 */

unsigned int
ust_get_tag_len(ust_ctx_t *ctx);
```