GOST Cipher Suites for TLS 1.2
draft-smyshlyaev-tls12-gost-suites-01

Abstract

This document specifies a set of cipher suites for the Transport Layer Security (TLS) protocol Version 1.2 to support the Russian cryptographic standard algorithms.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on December 31, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
1. Introduction ........................................... 3
2. Conventions Used in This Document ......................... 3
3. Basic Terms and Definitions ............................... 3
4. Cipher Suite Definitions .................................. 4
4.1. Record Payload Protection ............................... 4
4.1.1. CTR_OMAC ....................................... 5
4.1.2. CNT_IMIT ......................................... 6
4.2. Key Exchange and Authentication ......................... 6
4.2.1. Hello Messages ..................................... 8
4.2.1.1. Signature Algorithms Extension ................ 9
4.2.2. CertificateRequest .................................. 9
4.2.3. ClientKeyExchange .................................. 10
4.2.3.1. CTR_OMAC .................................... 11
4.2.3.2. CNT_IMIT ..................................... 12
4.2.4. CertificateVerify ................................... 14
4.3. Cryptographic Algorithms ............................... 14
4.3.1. Block Cipher ....................................... 14
4.3.2. MAC ............................................... 15
4.3.3. Encryption Algorithm ............................... 15
4.3.4. SNMAX ............................................ 15
4.3.5. PRF and HASH ...................................... 15
5. Additional Algorithms ..................................... 16
5.1. TLSTREE .............................................. 16
5.1.1. Key Tree Parameters .................................. 16
5.2. KExp15 and KImp15 Algorithms ......................... 17
5.3. KEG Algorithm ........................................ 18
5.4. gostIMIT28147 ......................................... 19
6. IANA Considerations ....................................... 19
7. Security Considerations ................................... 19
8. References ............................................... 19
8.1. Normative References ................................... 19
8.2. Informative References .................................. 21
Appendix A. Test Examples .................................... 22
A.1. Test Examples for CTR_OMAC cipher suites .............. 22
A.2. Test Examples for CNT_IMIT cipher suites .............. 22
Appendix B. Contributors .................................... 22
Appendix C. Acknowledgments .................................. 22
Authors’ Addresses .......................................... 22
1. Introduction

This document specifies three new cipher suites for the Transport Layer Security (TLS) Protocol Version 1.2 [RFC5246] to support the set of Russian cryptographic standard algorithms (called GOST algorithms). All of them use the GOST R 34.11-2012 [GOST3411-2012] hash algorithm (the English version can be found in [RFC6986]) and the GOST R 34.10-2012 [GOST3410-2012] signature algorithm (the English version can be found in [RFC7091]) but use different encryption algorithms, so they are divided into two types: the CTR_OMAC cipher suites and the CNT_IMIT cipher suite.

The CTR_OMAC cipher suites use the GOST R 34.12-2015 [GOST3412-2015] block ciphers (the English version can be found in [RFC7801]):

- TLS_GOSTR341112_256_WITH_KUZNYECHIK_CTR_OMAC = {0xXX, 0xXX};
- TLS_GOSTR341112_256_WITH_MAGMA_CTR_OMAC = {0xXX, 0xXX}.

The CNT_IMIT cipher suites use the GOST 28147-89 [GOST28147-89] block cipher (the English version can be found in [RFC5830]):

- TLS_GOSTR341112_256_WITH_28147_CNT_IMIT = {0xXX, 0xXX}.

2. Conventions Used in This Document

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 [RFC2119].

3. Basic Terms and Definitions

This document uses the following terms and definitions for the sets and operations on the elements of these sets:

- \( B^* \) the set of all byte strings of a finite length (hereinafter referred to as strings), including the empty string;
- \( B_s \) The set of byte vectors of size \( s \), \( s \geq 0 \), for \( s = 0 \) the \( B_s \) set consists of a single empty element of size 0. If \( W \) is an element of \( B_s \), then \( W = (w^1, w^2, \ldots, w^s) \), where \( w^1, w^2, \ldots, w^s \) are in \( \{0, \ldots, 255\} \);
- \( b[i..j] \) the string \( b[i..j] = (b_i, b_{i+1}, \ldots, b_j) \) in \( B_{(j-i+1)} \) where \( 1 \leq i \leq j \leq s \) and \( b = (b_1, \ldots, b_s) \) in \( B_s \);
- \( |X| \) the byte length of the byte string \( X \);
A | C  concatenation of strings A and C both belonging to B*, i.e.,
a string in B_(A+C), where the left substring in B_A is
equal to A, and the right substring in B_C is equal to C;

Int_s  the transformation that maps a string a = (a_s, ..., a_1) in
B_s into the integer Int_s(a) = 256^{s-1} * a_s + ... + 256 *
a_2 + a_1 (the interpretation of the binary string as an
integer);

Vec_s  the transformation inverse to the mapping Int_s (the
interpretation of an integer as a binary string);

Str_s  the transformation that maps an integer i = 256^{s-1} * i_s +
... + 2 * i_2 + i_1 into the string Str_s(i) = (i_1, ..., i_s) in B_s;

k  the byte-length of the block cipher key;

n  the block size of the block cipher (in bytes);

Q_C  the public key stored in the client’s certificate;

k_C  the private key that corresponds to Q_C key;

Q_S  the server’s public key;

k_C  the server’s private key;

r_C  the random string that corresponds to ClientHello.random
field from [RFC5246];

r_S  the random string that corresponds to ServerHello.random
field from [RFC5246].

4. Cipher Suite Definitions

4.1. Record Payload Protection

All of the cipher suites described in this document MUST use the
stream cipher (see Section 4.3.3) to protect records. The
TLSCiphertext structure for the CTR_OMAC and CNT_IMIT cipher suits is
specified in accordance with the GenericStreamCipher case (see
Section 6.2.3.1 of [RFC5246]):
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    GenericStreamCipher fragment;
} TLSCiphertext;

Here TLSCiphertext.fragment is generated as described in Section 4.1.1 and Section 4.1.2.

The connection key material consists of the sender_write_key (either the client_write_key or the server_write_key), the sender_write_MAC_key (either the client_write_MAC_key or the server_write_MAC_key) and the sender_write_IV (either the client_write_IV or the server_write_IV) parameters that are generated according to section 6.3 of [RFC5246].

4.1.1. CTR_OMAC

In case of the CTR_OMAC cipher_suites there is a certain key material which is used by the peer to protect a record that corresponds to the seqnum sequence number (in the following simply record key material) and consist of K_ENC_seqnum, K_MAC_seqnum and IV_seqnum values that are calculated using the TLSTREE function defined in Section 5.1 and the connection key material:

\[
K_{\text{ENC}}_{\text{seqnum}} = \text{TLSTREE}(\text{sender_write_key}, \text{seqnum});
\]

\[
K_{\text{MAC}}_{\text{seqnum}} = \text{TLSTREE}(\text{sender_write_MAC_key}, \text{seqnum});
\]

\[
IV_{\text{seqnum}} = \text{Vec}_{n/2}((\text{Int}_{n/2}(\text{sender_write_IV}) + \text{seqnum}) \mod 2^{n*8/2}).
\]

The TLSCiphertext.fragment that corresponds to the seqnum sequence number is equal to the ENCValue_seqnum value that is calculated as follows:

1. The MAC value (MACValue_seqnum) is generated by the MAC algorithm (see Section 4.3.2) similar to Section 6.2.3.1 of [RFC5246] except the used MAC key: the sender_write_MAC_key is replaced by the K^seqnum_MAC key:

\[
\text{MACData}_{\text{seqnum}} = \text{Str}_8(\text{seqnum}) | \text{type}_{\text{seqnum}} | \text{version}_{\text{seqnum}} | \text{length}_{\text{seqnum}} | \text{fragment}_{\text{seqnum}};
\]

\[
\text{MACValue}_{\text{seqnum}} = \text{MAC}(K_{\text{MAC}}_{\text{seqnum}}, \text{MACData}_{\text{seqnum}}),
\]
where type_seqnum, version_seqnum, length_seqnum, fragment_seqnum are the TLSCompressed.type, the TLSCompressed.version, the TLSCompressed.length and the TLSCompressed.fragment values of the record with the seqnum sequence number.

2. The stream cipher ENC (see Section 4.3.3) encrypts the entire data with the MACValue as follows:

\[ \text{ENCData_seqnum} = \text{fragment_seqnum} \mid \text{MACValue_seqnum}; \]
\[ \text{ENCValue_seqnum} = \text{ENC}(K_{\text{ENC}}_{\text{seqnum}}, \text{IV}_{\text{seqnum}}, \text{ENCData}_{\text{seqnum}}). \]

4.1.2. CNTIMIT

In case of the CNTIMIT cipher_suites the TLSCiphertext.fragment that corresponds to the seqnum sequence number is equal to the ENCValue_seqnum value that is calculated as follows:

1. The MAC value (MACValue_seqnum) is generated by the MAC algorithm (see Section 4.3.2) as follows:

\[ \text{MACData}_i = \text{Str}_8(i) \mid \text{type}_i \mid \text{version}_i \mid \text{length}_i \mid \text{fragment}_i, \]
\[ i \in \{0, \ldots, \text{seqnum}\}; \]
\[ \text{MACValue_seqnum} = \text{MAC}(\text{sender\_write\_MAC\_key}, \text{MACData}_0 \mid \ldots \mid \text{MACData}_{\text{seqnum}}), \]

where type_i, version_i, length_i, fragment_i are the TLSCompressed.type, the TLSCompressed.version, the TLSCompressed.length and the TLSCompressed.fragment values of the record with the i sequence number.

2. The stream cipher ENC (see Section 4.3.3) encrypts the entire data ENCData_i with the MACValue_i that correspond to the record sequence number i with the MACValue as follows:

\[ \text{ENCData}_i = \text{fragment}_i \mid \text{MACValue}_i, \ i \in \{0, \ldots, \text{seqnum}\}; \]
\[ \text{ENCValue}_0 \mid \ldots \mid \text{ENCValue}_{\text{seqnum}} = \text{ENC}(\text{sender\_write\_key}, \text{sender\_write\_IV}, \text{ENCData}_0 \mid \ldots \mid \text{ENCData}_{\text{seqnum}}). \]

4.2. Key Exchange and Authentication

All of the cipher suites described in this document use ECDH to compute the TLS premaster secret.
Figure 1 shows all messages involved in the TLS key establishment protocol (aka full handshake). A ServerKeyExchange MUST NOT be sent (the server’s certificate contains all the necessary keying information required by the client to arrive at the premaster secret).

The key exchange process consists of the following steps:

1. The client generates ECDHE key pair (Q_eph, k_eph), Q_eph is on the same curve as the server’s long-term public key Q_S.

2. The client generates the premaster secret value PS. The PS value is chosen by random from B_32.

3. Using k_eph, server long-term public key and some generated nonce value the client generates the encryption key for key-wrap algorithm and then sends the PS value wrapped with particular key-wrap algorithm.

4. The client sends its ephemeral public key Q_eph and the wrapped PS value in the ClientKeyExchange message.

5. The server extract the premaster secret value PS using its long-term secret key k_S in accordance with the key wrap algorithm.

The server side of the channel is always authenticated; the client side is optionally authenticated. The server is authenticated using
it’s long term private key from the certificate and proving that it
knows a shared secret. The client is authenticated when the server
is checking its signature.

The proposed cipher suites has direct impact only on the ClientHello,
the ServerHello, the CertificateRequest, the ClientKeyExchange and
the CertificateVerify handshake messages, that are described bellow
in greater detail in terms of the content and processing of these
messages.

4.2.1. Hello Messages

The ClientHello message must meet the following requirements:

- The ClientHello.compression_methods field MUST contain exactly one
  byte, set to zero, which corresponds to the "null" compression
  method.

- While using cipher CTR_OMAC cipher suites the
  ClientHello.extensions field MUST contain the following three
  extensions: signature_algorithms (see Section 4.2.1.1),
  extended_master_secret (see [RFC7627]), renegotiation_info (see
  [RFC5746]).

- While using the CNTIMIT cipher suite the ClientHello.extensions
  field MUST contain the signature_algorithms (see Section 4.2.1.1)
  extension. And it is RECOMMENDED to contain the following two
  extensions: extended_master_secret (see [RFC7627]),
  renegotiation_info (see [RFC5746]).

The ServerHello message must meet the following requirements:

- The ServerHello.compression_method field MUST contain exactly one
  byte, set to zero, which corresponds to the "null" compression
  method.

- While using the CTR_OMAC cipher suites the ServerHello.extensions
  field MUST contain the following two extensions:
  extended_master_secret (see [RFC7627]), renegotiation_info (see
  [RFC5746]).

- While using the CNTIMIT cipher suite it is RECOMMENDED for the
  ServerHello.extensions field to contain the following two
  extensions: extended_master_secret (see [RFC7627]),
  renegotiation_info (see [RFC5746]).

Note: If the extended_master_secret extension is agreed, then the
master secret value MUST be calculated in accordance with [RFC7627].
4.2.1.1. Signature Algorithms Extension

The signature_algorithms extension is described in Section 7.4.1.4.1 of (see [RFC5246]) and is specified as follows.

```c
SignatureAndHashAlgorithm
 supported_signature_algorithms<2..2^16-2>;

struct {
    HashAlgorithm hash;
    SignatureAlgorithm signature;
} SignatureAndHashAlgorithm;
```

The set of supported hash algorithms is specified as follows:

```c
enum {
    gostr3412012_256(238),
    gostr3412012_512(239),
} HashAlgorithm;
```

where gostr3412012_256 and gostr3412012_512 values correspond to the GOST R 34.11-2012 [GOST3411-2012] hash algorithm with 32-byte (256-bit) and 64-byte (512-bit) hash code respectively.

The set of supported signature algorithms is specified as follows:

```c
enum {
    gostr34102012_256(238),
    gostr34102012_512(239),
} SignatureAlgorithm;
```

where gostr34102012_256 and gostr34102012_512 values correspond to the GOST R 34.10-2012 [GOST3410-2012] signature algorithm with 32-byte (256-bit) and 64-byte (512-bit) key length respectively.

4.2.2. CertificateRequest

When this message is sent: this message is sent when requesting client authentication.

Meaning of this message: the server uses this message to suggest acceptable certificates.
The TLS CertificateRequest message is extended as follows.

```c
struct {
    ClientCertificateType certificate_types<1..2^8-1>;
    SignatureAndHashAlgorithm
        supported_signature_algorithms<2..2^16-2>;
    DistinguishedName certificate_authorities<0..2^16-1>;
} CertificateRequest;
```

where the SignatureAndHashAlgorithm structure is specified in Section 4.2.1.1, the ClientCertificateType and the DistinguishedName structures are specified as follows.

```c
enum {
    gostr34102012_256(238),
    gostr34102012_512(239), (255)
} ClientCertificateType;
```

opaque DistinguishedName<1..2^16-1>;

### 4.2.3. ClientKeyExchange

The ClientKeyExchange structure is defined as follows.

```c
enum { vko_kdf_gost, vko_gost } KeyExchangeAlgorithm;
```

```c
struct {
    select (KeyExchangeAlgorithm) {
        case vko_kdf_gost: PSKeyTransport;
        case vko_gost: TLSGostKeyTransportBlob;
    } exchange_keys;
} ClientKeyExchange;
```

The PSKeyTransport structure corresponds to the CTR_OMAC cipher suites and is described in Section 4.2.3.1 and the TLSGostKeyTransportBlob corresponds to CNTIMIT cipher suite and is described in Section 4.2.3.2.
4.2.3.1. CTR_OMAC

The CTR_OMAC cipher suites use the KExp15 and the KImp15 algorithms defined in Section 5.2 for key wrapping.

The export representation of the PS value is calculated as follows.

1. The client generates the keys $K^{\text{EXP\_MAC}}$ and $K^{\text{EXP\_ENC}}$ using the KEG function described in Section 5.3:

   $$H = \text{HASH}(r_C \mid r_S);$$

   $$K^{\text{EXP\_MAC}} \mid K^{\text{EXP\_ENC}} = \text{KEG}(k_{eph}, Q_S, H).$$

2. The client generates export representation of the premaster secret value PS:

   $$IV = H[25..24 + n / 2];$$

   $$\text{PSExp} = \text{KExp15}(PS, K^{\text{EXP\_MAC}}, K^{\text{EXP\_ENC}}, IV).$$

3. The client creates the PSKeyTransport structure that is defined as follows:

   PSKeyTransport ::= SEQUENCE {
     PSEXP OCTET STRING,
     ephemeralPublicKey SubjectPublicKeyInfo
   }

   SubjectPublicKeyInfo ::= SEQUENCE {
     algorithm AlgorithmIdentifier,
     subjectPublicKey BITSTRING
   }

   AlgorithmIdentifier ::= SEQUENCE {
     algorithm OBJECT IDENTIFIER,
     parameters ANY OPTIONAL
   }

Here the PSEXP field contains the PSExp value and the ephemeralPublicKey field contains the $Q_{eph}$ value.

After receiving the ClientKeyExchange message the server process it as follows.

1. Checks the next three conditions fulfilling and terminates the connection with fatal error if not.
2. Generates the keys $K^{\text{EXP\_MAC}}$ and $K^{\text{EXP\_ENC}}$ using the KEG function described in Section 5.3:

$$H = \text{HASH}(r_C \mid r_S);$$

$$K^{\text{EXP\_MAC}} \mid K^{\text{EXP\_ENC}} = \text{KEG}(k_S, Q_{\text{eph}}, H).$$

3. Extracts the common secret $PS$ from the export representation $PSExp$:

$$IV = H[25..24+n/2];$$

$$PS = \text{KImp15}(PSExp, K^{\text{EXP\_MAC}}, K^{\text{EXP\_ENC}}, IV).$$

4.2.3.2. CNT_IMIT

The client generates the key $KEK$ using the VKO function. VKO is one of the functions VKO_GOSTR3410_2012_256 or VKO_GOSTR3410_2012_512 described in [RFC7836]. The particular function depends on the elliptic curve used in the server certificate. The KEK calculation is made as follow

$$UKM = \text{HASH}(r_C \mid r_S)$$

$$KEK = \text{VKO}(k_{\text{eph}}, Q_S, UKM)$$

Generates the diversified key $KEK(UKM)$ using the function CryptoPro KEK Diversification Algorithm defined in [RFC4357]

Generates export representation of the common secret $PS$:

Compute a 4-byte MAC value, gost28147IMIT (UKM, KEK(UKM), CEK) as described in Section 5.4. Call the result $CEK$ MAC.

Encrypt $CEK$ in ECB mode using $KEK(UKM)$. Call the ciphertext $CEK$ ENC.

The wrapped key is the string $UKM \mid CEK$ ENC $\mid CEK$ MAC in B_44.
The Gost28147-89-EncryptedKey.encryptedKey value contains CEK_ENC value, the Gost28147-89-EncryptedKey.macKey contains CEK_MAC, and GostR3410-TransportParameters.ukm contains the UKM value.

There MUST be a keyBlob.transportParameters.ephemeralPublicKey field containing the client ephemeral public key Q_eph.

After receiving ClientKeyExchange message server process it as follows

- Checks the next four conditions fulfilling and terminates the connection with fatal error if not.
  1. Q_eph is on the same curve as server public key;
  2. Q_eph is not equal to zero point;
  3. q * Q_eph is not equal to zero point;
  4. Checks if UKM = HASH(r_C | r_S).

- Generates the key KEK using the VKO function.
  
  KEK = VKO(k_S, Q_eph, UKM).

- Generates the diversified key KEK(UKM) using the function CryptoPro KEK Diversification Algorithm defined in [RFC4357]
Internet-DraGOST Cipher Suites for Transport Layer Security (June 2018)

- Extracts the common secret PS from the export representation:
  - Decrypt CEK_ENC in ECB mode using KEK(UKM). Call the result CEK.
  - Compute a 4-byte MAC value, gost28147IMIT(UKM, KEK(UKM), CEK).
  - If the result is not equal to CEK_MAC return a fault value.

4.2.4. CertificateVerify

The TLS CertificateVerify message is extended as follows.

```plaintext
struct {
  SignatureAndHashAlgorithm algorithm;
  opaque signature<0..2^16-1>;
} CertificateVerify;
```

where SignatureAndHashAlgorithm structure is specified in Section 4.2.1.1.

The CertificateVerify.signature field is specified as follows.

```plaintext
CertificateVerify.signature = SIGN_{k_C}(handshake_messages) = Str_l(r) | Str_l(s)
```

where SIGN_{k_C} is the GOST R 34.10-2012 [GOST3410-2012] signature algorithm, k_C is a client long-term private key that corresponds to the client long-term public key Q_C from the client’s certificate, l = 32 for gostr34102012_256 signature algorithm and l = 64 for gostr34102012_512 signature algorithm.

4.3. Cryptographic Algorithms

4.3.1. Block Cipher

The cipher suite TLS_GOSTR341112_256_WITH_KUZNYECHIK_CTR_OMAC MUST use Kuznyechik [RFC7801] as a base block cipher for the encryption and MAC algorithm. The block length for this suite is 16 bytes and the key length is 32 bytes. The cipher suite TLS_GOSTR341112_256_WITH_MAGMA_CTR_OMAC MUST use Magma [GOST3412-2015] as a base block cipher for the encryption and MAC algorithm. The block length for this suite is 8 bytes and the key length is 32 bytes.
The cipher suite TLS_GOSTR341112_256_WITH_28147_CNT_IMIT MUST use GOST 28147-89 as a base block cipher [RFC5830] with the set of parameters id-tc26-gost-28147-param-Z defined in [RFC7836]. The block length for this suite is 8 bytes and the key length is 32 bytes.

4.3.2. MAC

The CTR_OMAC cipher suites use the OMAC message authentication code construction defined in [GOST3413-2015], which can be considered as the CMAC mode defined in [RFC4493] where Kuznyechik or Magma block cipher (see Section 4.3.1) are used instead of AES block cipher (see [IK2003] for more detail) as the MAC function.

The CNT_IMIT cipher suite uses the message authentication code function gostIMIT28147 defined in Section 5.4 with the initialization vector IV = IV0, where IV0 in B_8 is a string of all zeros, with the CryptoPro Key Meshing algorithm defined in [RFC4357].

4.3.3. Encryption Algorithm

The CTR_OMAC cipher suites use the block cipher in CTR-ACPKM encryption mode defined in [DraftRekeying] as the ENC encryption algorithm. The section size N is 4 KB for TLS_GOSTR341112_256_WITH_KUZNYECHIK_CTR_OMAC cipher suite and 1 KB for TLS_GOSTR341112_256_WITH_MAGMA_CTR_OMAC cipher suite. The initial counter nonce is defined as in Section 4.1.

The CNT_IMIT cipher suite uses the block cipher in counter encryption mode (CNT) defined in Section 6 of [RFC5830] with the CryptoPro Key Meshing algorithm defined in [RFC4357] as the ENC encryption algorithm.

4.3.4. SNMAX

The SNMAX parameter defines the maximal amount of messages that can be send during one TLS 1.2 connection. For TLS_GOSTR341112_256_WITH_KUZNYECHIK_CTR_OMAC cipher suite this amount is 2^64 - 1 messages and for TLS_GOSTR341112_256_WITH_MAGMA_CTR_OMAC is 2^32 - 1 messages.

4.3.5. PRF and HASH

The pseudorandom function (PRF) for all the cipher suites defined in this document is the PRF_TLS_GOSTR3411_2012_256 function described in [RFC7836].
The hash function Hash for all the cipher suites defined in this document is the GOST R 34.11-2012 [GOST3411-2012] hash algorithm with 32-byte (256-bit) hash code.

5. Additional Algorithms

5.1. TLSTREE

The TLSTREE function is defined as follows:

\[
\text{TLSTREE}(K_{\text{root}}, i) = KDF_3(KDF_2(KDF_1(K_{\text{root}}, \text{Vec}(i \& C_1)), \text{Vec}(i \& C_2)), \text{Vec}(i \& C_2)),
\]

where

- $K_{\text{root}}$ in $\mathbb{B}_{32}$;
- $i$ in $\{0, 1, \ldots, 2^{64} - 1\}$;
- $C_1, C_2, C_3$ are constants defined by the particular cipher suite (see Section 5.1.1);
- $KDF_j(K, D)$, $j = 1, 2, 3$, $K$ in $\mathbb{B}_{32}$, $D$ in $\mathbb{B}_8$, is the key derivation functions based on the KDF_GOSTR3411_2012_256 function defined in [RFC7836]:

\[
\begin{align*}
KDF_1(K, D) &= \text{KDF_GOSTR3411_2012_256}(K, \"level1", D);
KDF_2(K, D) &= \text{KDF_GOSTR3411_2012_256}(K, \"level2", D);
KDF_3(K, D) &= \text{KDF_GOSTR3411_2012_256}(K, \"level3", D).
\end{align*}
\]

5.1.1. Key Tree Parameters

The CTR_OMAC cipher suites use the TLSTREE function for the re-keying approach. The constants for it are defined as in the table below.
### Key tree constants

<table>
<thead>
<tr>
<th>CipherSuites</th>
<th>C_1, C_2, C_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS_GOSTR341112_256_WITH_KUZNYECHIK_CTR_</td>
<td>C_1=0xFFFFFFFF00000000,</td>
</tr>
<tr>
<td>OMAC</td>
<td>C_2=0xFFFFFFFFFFFFFF8000,</td>
</tr>
<tr>
<td></td>
<td>C_3=0xFFFFFFFFFFFFC0</td>
</tr>
<tr>
<td>TLS_GOSTR341112_256_WITH_MAGMA_CTR_OMAC</td>
<td>C_1=0xFFFFFFC0000000000,</td>
</tr>
<tr>
<td></td>
<td>C_2=0xFFFFFFFFFEE000000,</td>
</tr>
<tr>
<td></td>
<td>C_3=0xFFFFFFFFFFFFF0000</td>
</tr>
</tbody>
</table>

#### 5.2. KExp15 and KImp15 Algorithms

Algorithms KExp15 and KImp15 are keywrap algorithms that provide confidentiality and integrity of keys. These algorithms use the block cipher defined by the particular cipher suite.

The inputs of Kexp15 key export algorithm are

- key K in $V^*$;
- MAC key $K^{\text{Exp_MAC}}$ in $B_k$;
- encryption key $K^{\text{Exp_ENC}}$ in $B_k$;
- IV value in $B_{(n/2)}$.

The keys $K^{\text{Exp_MAC}}$ and $K^{\text{Exp_ENC}}$ MUST be independent. The export representation of the key K is computed as follows:

- Compute the MAC value of n byte length:
  $\text{KEYMAC} = \text{OMAC}(K^{\text{Exp_MAC}}, IV \mid K)$,
  where $\text{OMAC}(K, M)$ is a MAC function defined in Section 4.3.2.

- Compute the KEXP value:
  $\text{KEXP} = \text{encKey} \mid \text{encKeyMac} = \text{CTR}(K^{\text{Exp_ENC}}, IV, \text{KEYMAC})$,
  where $|\text{encKey}| = |K|$, $|\text{encKeyMac}| = |\text{KEYMAC}|$, $\text{CTR}(K, IV, M)$ is the counter encryption mode CTR defined in [GOST3413-2015] where $s = n$ which can be considered as the CTR mode defined in [MODES].

- The export representation of key K is the result of the Kexp15 algorithm and is defined as
  $\text{KExp15}(K, K^{\text{Exp_MAC}}, K^{\text{Exp_ENC}}, IV) = \text{KEXP}$. 
The import of key $K$ via $K^{imp15}$ algorithm is restoring the key $K$ from export representation $K^{exp}$ with keys $K^{exp\_MAC}$ and $K^{exp\_ENC}$ from $B_k$ and value IV from $B_{(n/2)}$. This is performed as follows:

- The string $K^{exp}$ is decrypted on the key $K^{exp\_ENC}$ with counter encryption mode CTR. The result of this operation is the string $K|KEYMAC$.

- Compute the MAC value of $n$ byte length:
  
  $KEYMAC' = OMAC(K^{exp\_MAC}, IV \mid K)$.  

- If $KEYMAC$ is not equal to $KEYMAC'$ return a fault value.

- Otherwise the result of the $K^{imp15}$ algorithm is defined as $K^{imp15}(K^{exp}, K^{Kexp\_MAC}, K^{Kexp\_ENC}, IV)$ and is equal to string $K$.

During the use of one keypair $(K^{exp\_ENC}, K^{exp\_MAC})$ the IV values MUST be unique. For the import of key $K$ with the $K^{imp15}$ algorithm every IV value MUST be sent with the export key representation or be a preshared value.

### 5.3. KEG Algorithm

The KEG algorithm of export key elaboration takes on input private key $d$, public key $Q$ and string $h$ from $B_{32}$. Then it returns the string from $B_{64}$.

The KEG algorithm is defined by two distinct ways depending on the private key length.

If the length of private key $d$ is 64 bytes the KEG algorithm is defined as follows:

$$KEG(d, Q, h) = VKO_512(d, Q, UKM),$$

where $VKO_512$ is the $VKO\_GOSTR3410\_2012\_512$ function defined in [RFC7836] and the $UKM$ parameter is equal to $r = \text{Int}_32(h[1..16])$ if $r$ is not equal to 0 and is equal to 1 otherwise.

If the length of private key $d$ is 32 bytes the KEG algorithm is defined as follows:

$$KEG(d, Q, h) = KDFTREE_256(K_{\text{EXP}}, "kdf tree", \text{seed}, 1),$$

where $KDFTREE_256$ is the $KDF\_TREE\_GOSTR3411\_2012\_256$ function defined in [RFC7836] and the parameters $K_{\text{EXP}}$ and seed are defined as

$$K_{\text{EXP}} = VKO_256(d, Q, UKM);$$
UKM is equal to r if r is not equal to 0 and is equal to 1 otherwise;

\[ r = \text{Int}_32(h[1..16]); \]

\[ \text{seed} = h[17..24], \]

where VKO_256 is the function VKO_GOSTR3410_2012_256 defined in [RFC7836].

5.4. gostIMIT28147

gost28147IMIT (IV, K, M) IV in B_8, K in B_32, M in B* is a MAC computation algorithm with 4 bytes output that proceed as follow

1. Divide M into 8 byte blocks: \( M = M_0 | M_1 | \ldots | M_r \).
2. Let \( M' = M_0 \text{ (xor) IV} | M_1 | M_2 | \ldots | M_r \).
3. Compute MAC value with 4 byte length with algorithm described in [RFC5830] using K as key and M' as input.
4. The result of MAC computation is the result of gost28147IMIT (IV, K, M) algorithm.

6. IANA Considerations

IANA has added the following entries in the TLS Cipher Suite Registry: TODO

7. Security Considerations

8. References

8.1. Normative References

[DraftRekeying]


8.2. Informative References


Appendix A. Test Examples

A.1. Test Examples for CTR_OMAC cipher suites

A.2. Test Examples for CNT_IMIT cipher suites

Appendix B. Contributors

- Evgeny Alekseev
  CryptoPro
  alekseev@cryptopro.ru

- Ekaterina Smyshlyaeva
  CryptoPro
  ess@cryptopro.ru

- Grigory Sedov
  CryptoPro
  sedovgk@cryptopro.ru

- Dmitry Eremin-Solenikov
  Auriga
  dbaryshkov@gmail.com

Appendix C. Acknowledgments

Authors’ Addresses

Stanislav Smyshlyaev (editor)
CryptoPro
18, Suschevsky val
Moscow 127018
Russian Federation

Phone: +7 (495) 995-48-20
Email: svs@cryptopro.ru

Dmitry Belyavsky
Cryptocom
14/2 Kedrova st
Moscow 117218
Russian Federation

Email: beldmit@gmail.com