Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 11, 2018

S. Smyshlyaev, Ed.
E. Alekseev
E. Smyshlyaeva
G. Sedov
CryptoPro

June 9, 2018

GOST Cipher Suites for TLS 1.2
draft-smyshlyaev-tls12-gost-suites-00

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 11, 2018.

Table of Contents

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 >= 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, ..., w^s), where w^1, w^2, ..., w^s are in {0, ... , 255};

b[i..j] the string b[i..j] = (b_i, b_{i+1}, ... , b_j) in B_{j-i+1} where 1<=i<=j<=s and b = (b_1, ... , 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 general description of the TLSPlaintext, the TLSCompressed and the TLSCiphertext structures can be found in Sections 6.2.1, 6.2.2 и 6.2.3 of [RFC5246]. The TLSCiphertext structure for the CTR_OMAC and CNT_IMIT cipher suits is specified as follows.

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    opaque fragment[TLSCiphertext.length];
} TLSCiphertext;
```

The TLSCiphertext.fragment that corresponds to the seq_num record sequence number is formed as follows.

1. Generate the key material for the current record using the TLSTREE function defined in Section 5.1 and the session key material: sender_write_key (either the client_write_key or the server_write_key), sender_write_MAC_key (either the client_write_MAC_key or the server_write_MAC_key) and sender_write_IV (either the client_write_IV or the server_write_IV):

   K^{seq_num}_ENC = TLSTREE(sender_write_key, seq_num);

   K^{seq_num}_MAC = TLSTREE(sender_write_MAC_key, seq_num);

   IV_{seq_num} = Vec_{n/2}((Int_{n/2}(sender_write_IV) + seq_num) mod 2^{n*8/2}).

2. The MAC value (MACValue) 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^{seq_num}_MAC key:

   MACData = Str_8(seq_num)│TLSCompressed.type | TLSCompressed.version | TLSCompressed.length | TLSCompressed.fragment;

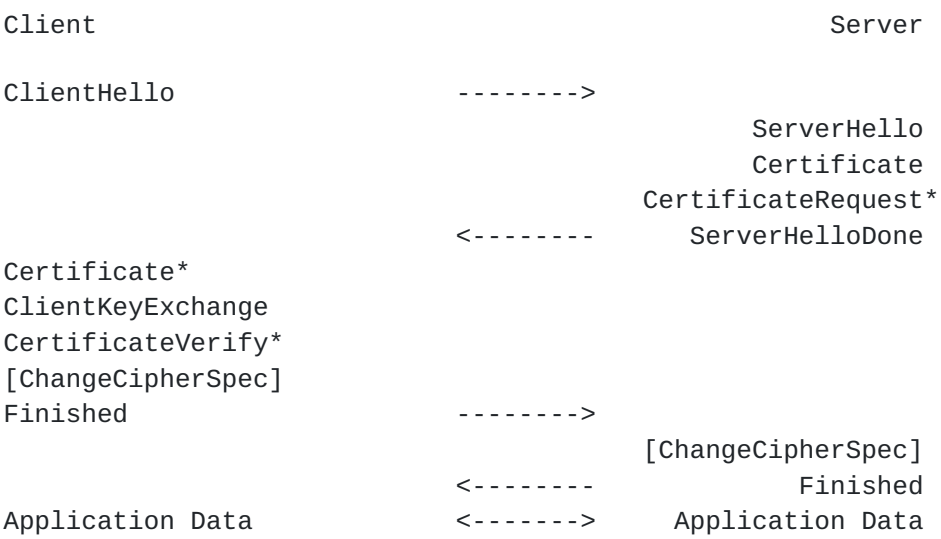
   MACValue = MAC(K^{seq_num}_MAC, MACData).

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

   TLSCiphertext.fragment = ENC(K^{seq_num}_ENC, IV_{seq_num}, TLSCompressed.fragment | MACValue).

4.2. Key Exchange and Authentication

All of the cipher suites described in this document use ECDH to compute the TLS premaster secret.



* message is not sent unless client authentication is desired

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 and server long-term public key 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 CNT_IMIT 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 CNT_IMIT 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.

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

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

The set of supported hash algorithms is specified as follows:

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

where gostr34112012_256 and gostr34112012_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:

```
enum {
    gostr34102012_256(238),
    gostr34102012_512(239), (255)
} 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.

```
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.

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

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

4.2.3. ClientKeyExchange

Client performs the following actions to create the ClientKeyExchange message:

- 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.
- Chooses randomly a premaster secret value PS from B_32;
- Creates the export representation of the PS value using some key wrap function.

The ClientKeyExchange structure is defined as follows.

```
enum { VKO_KDF_GOST, vko_gost } KeyExchangeAlgorithm;

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 CNT_IMIT 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^EXP_MAC and K^EXP_ENC using the KEG function described in Section 5.3:

   H = HASH(r_C | r_S);

   K^EXP_MAC | K^EXP_ENC = KEG(k_eph, Q_S, H).

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

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

   PSExp = KExp15(PS, K^EXP_MAC, K^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.

o Q_eph is on the same curve as server public key;

o Q_eph is not equal to zero point;

o q * Q_eph is not equal to zero point.

2. Generates the keys K^EXP_MAC and K^EXP_ENC using the KEG function described in Section 5.3:

```
H = HASH(r_C | r_S);

K^EXP_MAC | K^EXP_ENC = KEG(k_S, Q_eph, H).
```

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

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

PS = KImp15(PSExp, K^EXP_MAC, K^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 = HASH(r_C | r_S)

KEK = VKO(k_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 | CEK_ENC | CEK_MAC in B_44.

The TLSGostKeyTransportBlob is defined as

```
TLSGostKeyTransportBlob ::= SEQUENCE {
    keyBlob               GostR3410-KeyTransport,
}
GostR3410-KeyTransport ::=
    SEQUENCE {
        sessionEncryptedKey Gost28147-89-EncryptedKey,
        transportParameters [0] IMPLICIT GostR3410-TransportParameters OPTIONAL
}
Gost28147-89-EncryptedKey ::=
    SEQUENCE {
        encryptedKey Gost28147-89-Key,
        macKey Gost28147-89-MAC
}
GostR3410-TransportParameters ::=
    SEQUENCE {
        encryptionParamSet OBJECT IDENTIFIER,
        ephemeralPublicKey [0] IMPLICIT SubjectPublicKeyInfo OPTIONAL,
        ukm OCTET STRING
}
```

The Gost28147-89-EncryptedKey.encryptedKey value contais 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]

- 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.

```
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.

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

Both CTR_OMAC cipher suites use the MAC construction as defined in [GOST3413-2015].

The CNT_IMIT cipher suite uses the MAC construction defined in [RFC5830] with CryptoPro Key Meshing algorithm defined in [RFC4357] as follows. The MAC value MAC(K, M_t) for the message M_t is calculated with accordance to the next formula

MAC(K, M_t) = gostIMIT28147_MESH(IV0, K, M_0 | M_1 | ... | M_t)

Here gostIMIT28147_MESH(IV, K, M) is the gostIMIT28147(IV, K, M) function defined in Section 5.4 with CryptoPro Key Meshing algorithm defined in [RFC4357] and IV0 in B_8 is a string of all zeroes

4.3.3. Encryption Algorithm

Both CTR_OMAC cipher suites use the block cipher in CTR-ACPKM mode defined in [DraftRekeying]. The section size 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 counter mode construction defined in [RFC5830] with CryptoPro Key Meshing algorithm defined in [RFC4357]. The encryption of the record M_t is performed with accordance to the next formula

ENC(M_0) | ENC(M_1) | ... | ENC(M_t) = CNT_MESH(M_0 | M_1 | ... | M_t)

Here ENC(M_i) denotes the encryption of Record with number i and CNT_MESH denotes counter mode defined in [RFC5830] with CryptoPro Key Meshing algorithm defined in [RFC4357].

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. 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

CipherSuites	C_1, C_2, C_3
TLS_GOSTR341112_256_WITH_KUZNYECHIK_CTR_OMAC	C_1=0xFFFFFFFF00000000, C_2=0xFFFFFFFFFFF800, 00, C_3=0xFFFFFFFFFFFFFFC0
TLS_GOSTR341112_256_WITH_MAGMA_CTR_OMAC	C_1=0xFFFFFFC000000000, C_2=0xFFFFFFFFFE0000, 00, C_3=0xFFFFFFFFFFFFF000

4.3.6. 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:

```
TLSTREE(K_root, i) = KDF_3(KDF_2(KDF_1(K_root, Vec(i & C_1)),
Vec(i & C_2)), Vec(i & C_2))
```

where

- K_root in B_32;
- i in {0, 1, ... , 2^64 - 1};
- C_1, C_2, C_3 are constants defined by the particular cipher suite (see Section 4.3.5);
- KDF_j(K, D), j = 1, 2, 3, K in B_32, D in B_8, is the key derivation functions based on the KDF_GOSTR3411_2012_256 function defined in [RFC7836]:

```
KDF_1(K, D) = KDF_GOSTR3411_2012_256(K, "level1", D);
KDF_2(K, D) = KDF_GOSTR3411_2012_256(K, "level2", D);
KDF_3(K, D) = KDF_GOSTR3411_2012_256(K, "level3", D).
```

5.2. KExp15 and KImp15 Algorithms

Algorithms KExp15 and KImp15 are keywrap algorithms those 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^Exp_MAC in B_k

- Encryption key K^Exp_ENC in B_k
- IV value in B_{n/2}

The keys K^Exp_MAC and K^Exp_ENC MUST be independent. The export representation of the key K is computed as follows

- Compute the MAC value of n byte length

  KEYMAC = OMAC(K^Exp_MAC, IV | K)

  where OMAC(K, M) is a MAC function defined in [GOST3413-2015].

- Compute the KEXP value

  KEXP = encKey | encKeyMac = CTR(K^Exp_ENC, IV, KEYMAC)

  where |encKey| = |K|, |encKeyMAC| = |KEYMAC|, CTR(K, IV, M) is the counter encryption mode defined in [GOST3413-2015] where s = n.

- The export representation of key K is the result of algorithm Kexp15 and is defined as

  KExp15(K, K^Exp_MAC, K^Exp_ENC, IV) = KEXP.

The import of key K via KImp15 algorithm is restoring the key K from export representation KEXP 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 KEXP is decrypted on the key K^Exp_ENC with counter encryption mode defined in [GOST3413-2015] where s = n. The result of this operation is the string K|KEYMAC.
- Compute the MAC value of n byte length

  KEYMAC' = OMAC(K^Exp_MAC, IV | K).

- If KEYMAC is not equal to KEYMAC' return a fault value.
- Otherwise the result of the KImp15 algorithm is defined as KImp15(KEXP, 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 KImp15 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

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

where VKO_512 is the function VKO_GOSTR3410_2012_512 defined in [RFC7836] and the UKM parameter is equal to r = 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

KEG(d, Q, h) = KDFTREE_256(K_EXP, "kdf tree", seed, 1)

where KDFTREE_256 is the function KDF_TREE_GOSTR3411_2012_256 defined in [RFC7836] and the parameters K_EXP and seed are defined as

K_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 = Int_32(h[1..16])

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 | ... | M_r

2. Let M' = M_0 (xor) IV | M_1 | M_2 | ... | 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

TODO

8. References

8.1. Normative References

[DraftRekeying] Smyshlyaev, S., "Re-keying Mechanisms for Symmetric Keys", 2018, <https://tools.ietf.org/html/draft-irtf-cfrg-re-keying-12>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4357] Popov, V., Kurepkin, I., and S. Leontiev, "Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms", RFC 4357, DOI 10.17487/RFC4357, January 2006, <https://www.rfc-editor.org/info/rfc4357>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <https://www.rfc-editor.org/info/rfc5746>.

[RFC5830] Dolmatov, V., Ed., "GOST 28147-89: Encryption, Decryption, and Message Authentication Code (MAC) Algorithms", RFC 5830, DOI 10.17487/RFC5830, March 2010, <https://www.rfc-editor.org/info/rfc5830>.

[RFC6986] Dolmatov, V., Ed. and A. Degtyarev, "GOST R 34.11-2012: Hash Function", RFC 6986, DOI 10.17487/RFC6986, August 2013, <https://www.rfc-editor.org/info/rfc6986>.

[RFC7091] Dolmatov, V., Ed. and A. Degtyarev, "GOST R 34.10-2012: Digital Signature Algorithm", RFC 7091, DOI 10.17487/RFC7091, December 2013, <https://www.rfc-editor.org/info/rfc7091>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <https://www.rfc-editor.org/info/rfc7627>.

[RFC7801] Dolmatov, V., Ed., "GOST R 34.12-2015: Block Cipher "Kuznyechik"", RFC 7801, DOI 10.17487/RFC7801, March 2016, <https://www.rfc-editor.org/info/rfc7801>.

[RFC7836] Smyshlyaev, S., Ed., Alekseev, E., Oshkin, I., Popov, V., Leontiev, S., Podobaev, V., and D. Belyavsky, "Guidelines on the Cryptographic Algorithms to Accompany the Usage of Standards GOST R 34.10-2012 and GOST R 34.11-2012", RFC 7836, DOI 10.17487/RFC7836, March 2016, <https://www.rfc-editor.org/info/rfc7836>.

8.2. Informative References

[GOST28147-89]
Government Committee of the USSR for Standards, "Cryptographic Protection for Data Processing System, Gosudarstvennyi Standard of USSR (In Russian)", GOST 28147-89, 1989.

[GOST3410-2012]
Federal Agency on Technical Regulating and Metrology, "Information technology. Cryptographic data security. Signature and verification processes of [electronic] digital signature", GOST R 34.10-2012, 2012.

[GOST3411-2012]
Federal Agency on Technical Regulating and Metrology, "Information technology. Cryptographic Data Security. Hashing function", GOST R 34.11-2012, 2012.

[GOST3412-2015]
Federal Agency on Technical Regulating and Metrology, "Information technology. Cryptographic data security. Block ciphers", GOST R 34.12-2015, 2015.

[GOST3413-2015]
Federal Agency on Technical Regulating and Metrology, "Information technology. Cryptographic data security. Modes of operation for block ciphers", GOST R 34.13-2015, 2015.

Appendix A. Test Examples

A.1. Test Examples for TODO

A.2. Test Examples for TODO

Appendix B. Acknowledgments

We thank TODO for their useful comments.

Authors' Addresses

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

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

Evgeny Alekseev
CryptoPro
18, Suschevsky val
Moscow 127018
Russian Federation

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

Ekaterina Smyshlyaeva
CryptoPro
18, Suschevsky val
Moscow 127018
Russian Federation

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

Grigory Sedov
CryptoPro
18, Suschevsky val
Moscow 127018
Russian Federation

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