Network Working Group S. Smyshlyaev, Ed.
Internet-Draft V. Popov
Intended status: Informational E. Alekseev
Expires: December 25, 2015 I. Oshkin
CRYPTO-PRO
June 23, 2015

Guidelines on the cryptographic algorithms, accompanying the usage of standards GOST R 34.10-2012 and GOST R 34.11-2012
draft-smyshlyaev-gost-usage-00

Abstract

The usage of cryptographic algorithms, that are defined by GOST R 34.10-2012 [GOST3410-2012] and GOST R 34.11-2012 [GOST3411-2012] standards, for protection of the information is carried out, as a rule, within the cryptographic protocols based on the accompanying algorithms.

This memo contains a description of the accompanying algorithms for defining the pseudorandom functions, the key derivation functions, the key agreement protocols based on the Diffie-Hellman algorithm and the keying material export algorithms.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The usage of cryptographic algorithms, that are defined by the GOST R 34.10-2012 [GOST3410-2012] and GOST R 34.11-2012 [GOST3411-2012] standards, for protection of the information is carried out, as a rule, within the cryptographic protocols based on the accompanying algorithms.

The specifications of algorithms and parameters proposed in this memo are defined on the basis of experience in the development of cryptographic protocols, as described in the [RFC4357], [RFC4490] and [RFC4491].

This memo contains a description of the accompanying algorithms for defining the pseudorandom functions, the key derivation functions, the key agreement protocols based on the Diffie-Hellman algorithm and the keying material export algorithms.

This memo does not specify the cryptographic algorithms GOST R 34.10-2012 [GOST3410-2012] and GOST R 34.11-2012 [GOST3411-2012]. These algorithms are defined by the national standards GOST R 34.10-2012 [GOST3410-2012] and GOST R 34.11-2012 [GOST3411-2012] and described in [RFC7091] and [RFC6986] (an English version of Russian national standards).

The need to ensure compatibility of the cryptographic protocol implementations based on the Russian cryptographic standards GOST R 34.10-2012 [GOST3410-2012] and GOST R 34.11-2012 [GOST3411-2012] served as the main reason for the development of this document.

2. Scope

This memo is recommended for use in encrypting and protecting the authenticity of the data, based on the use of digital signature algorithms GOST R 34.10-2012 [GOST3410-2012] and hash function GOST R 34.11-2012 [GOST3411-2012], in public and corporate networks to protect information that does not contain a classified information.

3. 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].

3.1. Mathematical objects

This document uses the following notation for the sets and operations on the elements of these sets in accordance with GOST R 34.11-2012 [GOST3411-2012]:

(xor) exclusive-or of two binary vectors of the same length;

V_n the finite-dimensional vector space over GF(2) of dimension n with the (xor) operation, for n = 0 the V_0 space consists of a single empty element of size 0;

U the element of V_n; in the binary representation U = (u_(n-1), u_(n-2), ..., u_1, u_0), where u_i in {0, 1};

A|B concatenation of vectors A, B, i.e., if A in V_n1, B in V_n2, A = (a_(n1-1), a_(n1-2), ..., a_0), and B = (b_(n2-1), b_(n2-2), ..., b_0), then A|B = (a_(n1-1), a_(n1-2), ..., a_0, b_(n2-1), b_(n2-2), ..., b_0) is an element of V_(n1+n2);

V_(8, r) the set of byte strings of size r; if W is an element of V(8, r), then W = (w^0, w^1, ..., w^(r-1)), where w^0, w^1, ..., w^(r-1) are elements of V_8; if A in V_(8, r1), B in V_(8, r2), A = (a^0, a^1, ..., a^(r1-1)), and B = (b^0, b^1, ..., b^(r2-1)), then A|B = (a^0, a^1, ..., a^(r1-1), b^0, b^1, ..., b^(r2-1)) is an element of V_(8, r1+r2);

Bit representation the bit representation of the element W = (w^0, w^1, ..., w^(r-1)) of V_(8, r), where w^0 = (w_7, w_6, ..., w_0), w^1 = (w_15, w_14, ..., w_8), ..., w^(r-1) = (w_(8r-1), w_(8r-2), ..., w_(8r-8)) are elements of V_8, is an element (w_(8r-1), w_(8r-2), ..., w_1, w_0) of V_(8*r);

Byte representation if n is a multiple of 8, r = n/8, then the byte representation of the element W = (w_(n-1), w_(n-2), ..., w_0) of V_n is a byte string (w^0, w^1, ..., w^(r-1)) of V_(8, r), where w^0 = (w_7, w_6, ..., w_0), w^1 = (w_15, w_14, ..., w_8), ..., w^(r-1) = (w_(8r-1), w_(8r-2), ..., w_(8r-8)) are elements of V_8;

K (key) arbitrary element of V_n; if K in V_n, then its size (in bits) is equal to n, where n can be an arbitrary natural number.

Note: It is proposed to interpret and edit the formulas in accordance with the above definitions.

3.2. Basic terms and definitions

This memo uses the following terms, abbreviations and symbols:

Symbols	Meaning
H_256	GOST R 34.11-2012 hash function, 256-bit
H_512	GOST R 34.11-2012 hash function, 512-bit
HMAC	a function for calculating a message authentication code (based on some hash function)
HMAC_256	a function based on the hash function H_256, intended for computing a message authentication code
HMAC_512	a function based on the hash function H_512, intended for computing a message authentication code
PRF	a pseudorandom function, i.e., a transformation that allows to generate pseudorandom sequence of bytes
KDF	a key derivation function, i.e., a transformation, that allows to derive keys and keying material for the root key and random data using a pseudorandom function

To produce a byte sequence of the size N with functions that give a longer output the input should be taken from the output sequence of the first N bytes. This remark applies to the following functions:

- the functions described in Section 4.2;
- KDF_TREE_GOSTR3411_2012_256.

When n is multiple of 8, an element of V_n can be represented in the bit and byte form. The result of operation <<|>>, applied to the elements in the bit representation is described in the bit representation. The result of the operation <<|>>, applied to the same elements in their byte representation is described in the byte representation. Thus, the symbol <<|>> is used to refer to two different operations, depending on the form of their arguments. Selecting one of these operations is uniquely determined by the representation of arguments.

Hereinafter all data (the elements of V_n) unless otherwise specified, are considered given in the byte representation. Operation <<|>> on the arguments of functions, unless explicitly stated otherwise, is performed on their byte representation.

If the function is defined outside this document (eg, H_256) and its definition is using arguments in bit representation, it is assumed that the bit representation of the argument is formed immediately before the calculation of the function (in particular, only after the application of the operation <<|>> to the byte representation of the arguments).

If as the argument of the function defined below is used the output of another function that is defined outside of this document and has output value in bit representation, it is assumed that the output value will be translated into the byte representation before substitution in arguments.

4. Algorithm descriptions

For algorithms described in this paper, the possible values of the functions are limited by the permissibility of applying them as the input parameter of the transformations and are assigned by the protocols.

4.1. HMAC functions

This section defines the HMAC transformations based on GOST R 34.11-2012 [GOST3411-2012] algorithm with different size of the output value.

4.1.1. HMAC_GOSTR3411_2012_256

This HMAC transformation is based on GOST R 34.11-2012 [GOST3411-2012] algorithm, 256-bit output. The identifier of this transformation is shown below:

    id-tc26-hmac-gost-3411-12-256, <<1.2.643.7.1.1.4.1>>.

The calculation of HMAC_256(K, T) for the data T of arbitrary length and the key K of n bits size is the forming of the 64-byte string K* and the transformation on K* and T using the hash function H_256.

The size n can take any value in the interval from 256 to 512.

For the formation of the key K*: if n < 512, take the string K* equal to the byte representation of the bit string K | A, where A = (0, 0, ..., 0) in V_(512-n); if n = 512, take K* equal to the byte representation of the key K.

The value of HMAC_256 (K, T) is given by:

HMAC_256 (K, T) = H_256 (K* (xor) opad | H_256 (K* (xor) ipad | T)),

where byte representations are:

ipad = (0x36 | 0x36 | ... | 0x36) in V_(8, 64),
opad = (0x5C | 0x5C | ... | 0x5C) in V_(8, 64).

This algorithm uses H_256 as a hash function for HMAC, described in [RFC2104]. The method of forming the values of ipad and opad is also given in [RFC2104]. The size of the HMAC_256 output in bytes is equal to 32, the block size of the iterative procedure for the H_256 compression function in bytes is equal to 64 (in the notation of [RFC2104], L = 32 and B = 64, respectively).

4.1.2. HMAC_GOSTR3411_2012_512

This HMAC transformation is based on GOST R 34.11-2012 [GOST3411-2012], 512-bit output. The identifier of this transformation is shown below:

id-tc26-hmac-gost-3411-12-512, <<1.2.643.7.1.1.4.2>>.

The calculation of HMAC_512(K, T) for the data T of arbitrary length and the key K of n bits size is the forming of the 64-byte string K* and the transformation on K* and T using the hash function H_512.

The size n can take any value in the interval from 256 to 512. The recommended value is 512.

For the formation of the key K*: if n < 512, take the string K* equal to the byte representation of the bit string K | A, where A = (0, 0, ..., 0) in V_(512-n); if n = 512, take K* equal to the byte representation of the K key.

The value of HMAC_512 (K, T) is given by:

HMAC_512 (K, T) = H_512 (K* (xor) opad | H_512 (K* (xor) ipad | T)),

where byte representations are:

ipad = (0x36|0x36|...|0x36) in V_(8, 64),
opad = (0x5C|0x5C|...|0x5C) in V_(8, 64).

This algorithm uses H_512 as a hash function for HMAC, described in [RFC2104]. The method of forming the values of ipad and opad is also given in [RFC2104]. The size of the HMAC_512 output in bytes is equal to 64, the block size of the iterative procedure for the H_512 compression function in bytes is equal to 64 (in the notation of [RFC2104], L = 64 and B = 64, respectively).

4.2. PRF

This section defines six based on HMAC PRF transformations that are recommended for the use. Two of them are for the TLS protocol and four for IPsec.

To obtain a set of values of the total size of m bytes using any of the following PRF it should be taken equal to the corresponding sequential values from the first m bytes of the used PRF output in the byte representation.

4.2.1. PRFs for the TLS protocol

4.2.1.1. PRF_TLS_GOSTR3411_2012_256

This is the transformation to implement the pseudorandom function of the TLS protocol; the transformation uses the HMAC_256 values based on GOST R 34.11-2012 [GOST3411-2012], 256-bit output.

```
PRF_TLS_GOSTR3411_2012_256 (secret, label, seed) =
= P_GOSTR3411_2012_256 (secret, label | seed),

P_GOSTR3411_2012_256 (secret, S) =
= HMAC_256 (secret, A_1 | S) | HMAC_256 (secret, A_2 | S) |
HMAC_256 (secret, A_3 | S) | ...
```

The A_i parameters are determined sequentially as follows:

```
A_0 = S,
A_i = HMAC_256 (secret, A_(i-1)).
```

P_GOSTR3411_2012_256 function uses HMAC_256 and corresponds to the method of method of specifying the arguments and the output value of P_hash data expansion function, given in Section 5 of [RFC2246] and kept in [RFC5246].

4.2.1.2. PRF_TLS_GOSTR3411_2012_512

This is the transformation to implement the pseudorandom function of the TLS protocol; the transformation uses the HMAC_512 values based on GOST R 34.11-2012 [GOST3411-2012], 512-bit output.

```
PRF_TLS_GOSTR3411_2012_512 (secret, label, seed) =
= P_GOSTR3411_2012_512 (secret, label | seed),
```

```
P_GOSTR3411_2012_512 (secret, S) =
= HMAC_512 (secret, A_1 | S) | HMAC_512 (secret, A_2 | S) |
HMAC_512 (secret, A_3 | S) | ...
```

The A_i parameters are determined sequentially as follows:

```
A_0 = S,
A_i = HMAC_512 (secret, A_(i-1)).
```

P_GOSTR3411_2012_512 function uses HMAC_512 and corresponds to the method of method of specifying the arguments and the output value for P_hash data expansion function, given in Section 5 of [RFC2246] and kept in [RFC5246].

4.2.2. PRFs for the IPsec protocols based on GOST R 34.11-2012, 256-bit

4.2.2.1. PRF_IPSEC_KEYMAT_GOSTR3411_2012_256

This pseudorandom function for the keying material generation is defined as follows (the arguments are the byte strings K and S):

```
PRF_IPSEC_KEYMAT_GOSTR3411_2012_256 (K, S) = T1| T2| T3| T4|...,
```

where

```
T1 = HMAC_256 (K, S),
T2 = HMAC_256 (K, T1 | S),
T3 = HMAC_256 (K, T2 | S),
T4 = HMAC_256 (K, T3 | S),
...
```

PRF_IPSEC_KEYMAT_GOSTR3411_2012_256 function is similar to KEYMAT function in [RFC2409] regarding the assignment scheme for the arguments in the iterations.

4.2.2.2. PRF_IPSEC_PRFPLUS_GOSTR3411_2012_256

This pseudorandom function for the keying material generation is defined as follows (the arguments are the byte strings K and S)

```
PRF_IPSEC_PRFPLUS_GOSTR3411_2012_256 (K, S) = T1| T2| T3| T4|...,
```

where

```
T1 = HMAC_256 (K, S | 0x01),
T2 = HMAC_256 (K, T1 | S | 0x02),
T3 = HMAC_256 (K, T2 | S | 0x03),
T4 = HMAC_256 (K, T3 | S | 0x04),
```

```
...
```

PRF_IPSEC_PRFPLUS_GOSTR3411_2012_256 output size is not more than 255*256 bits, which corresponds to the output sequence T1| T2| T3| T4| ... | T255.

PRF_IPSEC_PRFPLUS_GOSTR3411_2012_256 is similar to the prf+ function in [RFC5996] regarding the assignment scheme for the arguments in iterations.

4.2.3. PRFs for the IPsec protocols based on GOST R 34.11-2012, 512-bit

4.2.3.1. PRF_IPSEC_KEYMAT_GOSTR3411_2012_512

This pseudorandom function for the keying material generation is defined as follows (the arguments are the byte strings K and S):

```
PRF_IPSEC_KEYMAT_GOSTR3411_2012_512 (K, S) = T1| T2| T3| T4|...,
```

where

```
T1 = HMAC_512 (K, S),
T2 = HMAC_512 (K, T1 | S),
T3 = HMAC_512 (K, T2 | S),
T4 = HMAC_512 (K, T3 | S),
...
```

PRF_IPSEC_KEYMAT_GOSTR3411_2012_512 is similar to KEYMAT function in [RFC2409] regarding the assignment scheme for the arguments in iterations.

4.2.3.2. PRF_IPSEC_PRFPLUS_GOSTR3411_2012_512

This pseudorandom function for the keying material generation is defined as follows (the arguments are the byte strings K and S):

```
PRF_IPSEC_PRFPLUS_GOSTR3411_2012_512 (K, S) = T1| T2| T3| T4|...,
```

where

```
T1 = HMAC_512 (K, S | 0x01),
T2 = HMAC_512 (K, T1 | S | 0x02),
T3 = HMAC_512 (K, T2 | S | 0x03),
T4 = HMAC_512 (K, T3 | S | 0x04),
...
```

PRF_IPSEC_PRFPLUS_GOSTR3411_2012_512 output size is not more than 255*512 bits, which corresponds to the output sequence T1| T2| T3| T4|...| T255.

The function PRF_IPSEC_PRFPLUS_GOSTR3411_2012_512 is similar to the prf+ function in [RFC5996] regarding the assignment scheme for the arguments in iterations.

4.3. VKO algorithms for key agreement

This section identifies the key agreement algorithms using GOST R 34.10-2012 [GOST3410-2012].

4.3.1. VKO_GOSTR3410_2012_256

The 256-bit VKO GOST R 34.10-2012 algorithm is used for an agreement of the VKO 256-bit keys and based on GOST R 34.11-2012 [GOST3411-2012], 256-bit. The algorithm can be used for an agreement of the GOST R 34.10-2012 [GOST3410-2012] keys with the size of 256 bits or 512 bits.

The algorithm is designed to produce an encryption key or a keying material of size 256 bits to be used in the cryptographic protocols. Key or keying material KEK_VKO (x, y, UKM) is produced by the exchange participant from his private key x, the public key y*P of the opposite side and UKM value, considered as a number.

The algorithm can be used for both static and ephemeral key with the public key size n >= 512 bits including the case where one side uses a static key and the other - ephemeral.

UKM parameter is optional (the default UKM = 1) and can take any value from 1 to 2^(n/2)-1. It is allowed to use a nonzero UKM of arbitrary size not exceeding n/2 bits. UKM size of 64 bit or more is recommended for cases where the keys at least one of the parties are static.

    K (x, y, UKM) = (m/q*UKM*x mod q)*(y*P),

where m and q - the parameters of the elliptic curve according GOST R 34.10-2012 [GOST3410-2012] notation.

    KEK_VKO (x, y, UKM) = H_256 (K(x, y, UKM)).

This algorithm is defined by analogy with Section 5.2 of [RFC4357], but instead of the hash function GOST R 34.11-94 [GOST3411-94] (referred as gostR3411) applies the hash function H_256 and K(x, y, UKM) is calculated at the public key size n >= 512 bits and UKM size up to n/2 bits.

4.3.2. VKO_GOSTR3410_2012_512

The 512-bit VKO GOST R 34.10-2012 algorithm is used for an agreement of the VKO 512-bit keys and based on GOST R 34.11-2012 [GOST3411-2012], 512-bit. The algorithm can be used for an agreement of the GOST R 34.10-2012 [GOST3410-2012] keys with the size of 512 bits.

The algorithm is designed to produce an encryption key or keying material of size 512 bits to be used in cryptographic protocols. Key or keying material KEK_VKO (x, y, UKM) is produced by the exchange participant from his private key x, the public key y*P of the opposite side and the UKM value, considered as a number.

The algorithm can be used for both static and ephemeral key with the public key size n >= 1024 bits including the case where one side uses a static key and the other - ephemeral.

UKM parameter is optional (the default UKM = 1) and can take any value from 1 to 2^(n/2)-1. It is allowed to use a nonzero UKM of arbitrary size not exceeding n/2 bits. UKM size of 128 bit or more is recommended for cases where the keys at least one of the parties are static.

K (x, y, UKM) = (m/q*UKM*x mod q)*(y*P),

where m and q - the parameters of the elliptic curve according GOST R 34.10-2012 [GOST3410-2012] notation.

KEK_VKO (x, y, UKM) = H_512 (K (x, y, UKM)).

This algorithm is defined by analogy with Section 5.2 of [RFC4357], but instead of the hash function GOST R 34.11-94 [GOST3411-94] (referred as gostR3411) applies the hash function H_256, and K(x, y, UKM) is calculated at the public key size n >= 1024 bits and UKM size up to n/2 bits.

4.4. Key derivation function KDF_GOSTR3411_2012_256

The key derivation function KDF_GOSTR3411_2012_256 based on HMAC_256 function is designed to generate a 256-bit keying material and is given by:

KDF (K_in, label, seed) = HMAC_256 (K_in, 0x01 | label | 0x00 | seed | 0x01 | 0x00), where

o K_in -- derivation key,

o label, seed -- the parameters, fixed and assigned by a protocol.

The key derivation function KDF_GOSTR3411_2012_256 is a special case of KDF_TREE_GOSTR3411_2012 function, described in the next section.

4.5. Key derivation function KDF_TREE_GOSTR3411_2012_256

The key derivation function KDF_TREE_GOSTR3411_2012_256 based on HMAC_256 and is given by:

KDF_TREE (K_in, label, seed, R) = K(1)| K(2)| K(3)| K(4)|...,

K(i) = HMAC_256 (K_in, [i]_2 | label | 0x00 | seed| [L]2), i >= 1,

where

R a fixed external parameter, with possible values of 1, 2, 3 or 4;

K_in derivation key;

L the required size (in bits) of the generated keying material (an integer, not exceeding 256*(2^(8*R)-1));

[L]_2 byte representation of L, in network byte order;

i iteration counter;

[i]_2 byte representation of the iteration counter (in the network byte order), the number of bytes in the representation [i]_2 is equal to R (no more than 4 bytes);

label, seed the parameters, fixed and assigned by a protocol.

The key derivation function KDF_TREE_GOSTR3411_2012_256 is intended for generating a keying material in size of L, not exceeding 256*(2^(8*R)-1) bits, and utilizes general principles of the input and output for the key derivation function that are outlined in Section 5.1 of NIST SP 800-108 [NISTSP800-108]. HMAC_256 algorithm with 256-bit output described in Section 4.1 is selected as a pseudorandom function.

When R = 1 and L = 256 the function KDF_TREE_GOSTR3411_2012_256 is equivalent to KDF_GOSTR3411_2012_256 from the previous section.

Each key derived from the keying material, which was formed with the derivation key K_in (0-level key) may be a 1-level diversification key and may used to generate a new keying material. The keying material derived from the 1-level derivation key, can be broken down into the 2nd level derivation keys. The application of this procedure leads to the construction of the key tree with the root key and the formation of the key material to the hierarchy of the levels, as described in Section 6 of NIST SP 800-108 [NISTSP800-108]. The partitioning procedure for keying material at each level is defined in the protocols.

4.6. Key wrap and unwrap

Wrapped representation of the secret key K (GOST R 34.10-2012 [GOST3410-2012] key or GOST 28147-89 [GOST28147-89] key) is formed as follows by using a given export key K_e (GOST 28147-89 [GOST28147-89] key) and the random UKM vector from 8 to 16 bytes in size:

1. Generates a random UKM vector.

2. With the key derivation function, using export key K_e as a derivation key, and a UKM vector as the value of seed, generates a key, denoted by KEK_e (UKM), where

   KEK_e (UKM) = KDF (K_e, label, UKM).

3. MAC value GOST 28147-89 (4-byte) for the data K and the key KEK_e (UKM) is calculated, initialization vector (IV) in this case is equal to the first 8 bytes of UKM. The resulting value is denoted as CEK_MAC.

4. The key K is encrypted by the GOST 28147-89 algorithm in the Electronic Codebook (ECB) mode with the key KEK_e (UKM). The encoding result denoted as CEK_ENC.

5. The wrapped representation of the key is considered (UKM | CEK_ENC | CEK_MAC).

Where as a key derivation function is used KDF function (see previous section) for the fixed value

   label = (0x26 | 0xBD | 0xB8 | 0x78)

and the seed value that is equal to UKM.

During key import the value of key K is restored as follows from the wrapped representation of the key (GOST R 34.10-2012 [GOST3410-2012] key or GOST 28147-89 key [GOST28147-89] key) and the export key K_e:

1. From the wrapped representation of the key selects the sets UKM, CEK_ENC, and CEK_MAC.

2. With the key derivation function, using the export key K_e as a derivation key, and a random UKM value as the value of seed, generates a key, denoted by KEK_e(UKM), where

   KEK_e (UKM) = KDF (K_e, label, UKM).

3. The CEK_ENC set is decrypted by the GOST 28147-89 algorithm in the Electronic Codebook (ECB) mode with the key KEK_e(UKM). The unwrapped key K is assumed to be equal to the result of decryption.

4. MAC value GOST 28147-89 (4-byte) for the data K and the key KEK_e(UKM) is calculated, initialization vector (IV) in this case is equal to the first 8 bytes of UKM. If the result does not equal to CEK_MAC, an error is returned.

The algorithms for wrapping and unwrapping of the GOST R 34.10-2012 [GOST3410-2012] keys are modifications of CryptoPro Key Wrap and CryptoPro Key Unwrap algorithms, described in Sections 6.3 and 6.4 of [RFC4357].

5. References

5.1. Normative References

[GOST28147-89]
Gosudarstvennyi Standard of USSR, Government Committee of the USSR for Standards, "Systems of information processing. Cryptographic data security. Algorithms of cryptographic transformation", 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.

[GOST3411-94]
Federal Agency on Technical Regulating and Metrology, "Information technology. Cryptographic Data Security. Hashing function", GOST R 34.11-94, 1994.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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, January 2006.

5.2. Informative References

[NISTSP800-108]
National Institute of Standards and Technology, "Recommendation for Key Derivation Using Pseudorandom Functions", NIST SP 800-108, October 2009.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC4490] Leontiev, S. and G. Chudov, "Using the GOST 28147-89, GOST R 34.11-94, GOST R 34.10-94, and GOST R 34.10-2001 Algorithms with Cryptographic Message Syntax (CMS)", RFC 4490, May 2006.

[RFC4491] Leontiev, S. and D. Shefanovski, "Using the GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms with the Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 4491, May 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.

[RFC6986] Dolmatov, V. and A. Degtyarev, "GOST R 34.11-2012: Hash Function", RFC 6986, August 2013.

[RFC7091] Dolmatov, V. and A. Degtyarev, "GOST R 34.10-2012: Digital Signature Algorithm", RFC 7091, December 2013.

Appendix A. Test examples

1) HMAC_GOSTR3411_2012_256

Key K:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
```

T:

```
01 26 bd b8 78 00 af 21 43 41 45 65 63 78 01 00
```

HMAC_256(K, T) value:

```
a1 aa 5f 7d e4 02 d7 b3 d3 23 f2 99 1c 8d 45 34
01 31 37 01 0a 83 75 4f d0 af 6d 7c d4 92 2e d9
```

2) HMAC_GOSTR3411_2012_512

Key K:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
```

T:

```
01 26 bd b8 78 00 af 21 43 41 45 65 63 78 01 00
```

HMAC_256(K, T) value:

```
a5 9b ab 22 ec ae 19 c6 5f bd e6 e5 f4 e9 f5 d8
54 9d 31 f0 37 f9 df 9b 90 55 00 e1 71 92 3a 77
3d 5f 15 30 f2 ed 7e 96 4c b2 ee dc 29 e9 ad 2f
3a fe 93 b2 81 4f 79 f5 00 0f fc 03 66 c2 51 e6
```

3) PRF_TLS_GOSTR3411_2012_256

Key K:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
```

Seed:

```
18 47 1d 62 2d c6 55 c4 d2 d2 26 96 91 ca 4a 56
0b 50 ab a6 63 55 3a f2 41 f1 ad a8 82 c9 f2 9a
```

Label:

```
11 22 33 44 55
```

Output T1:

```
ff 09 66 4a 44 74 58 65 94 4f 83 9e bb 48 96 5f
15 44 ff 1c c8 e8 f1 6f 24 7e e5 f8 a9 eb e9 7f
```

Output T2:

```
c4 e3 c7 90 0e 46 ca d3 db 6a 01 64 30 63 04 0e
c6 7f c0 fd 5c d9 f9 04 65 23 52 37 bd ff 2c 02
```

4) PRF_TLS_GOSTR3411_2012_512

Key K:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
```

Seed:

```
18 47 1d 62 2d c6 55 c4 d2 d2 26 96 91 ca 4a 56
0b 50 ab a6 63 55 3a f2 41 f1 ad a8 82 c9 f2 9a
```

Label:

```
11 22 33 44 55
```

Output T1:

```
f3 51 87 a3 dc 96 55 11 3a 0e 84 d0 6f d7 52 6c
5f c1 fb de c1 a0 e4 67 3d d6 d7 9d 0b 92 0e 65
ad 1b c4 7b b0 83 b3 85 1c b7 cd 8e 7e 6a 91 1a
62 6c f0 2b 29 e9 e4 a5 8e d7 66 a4 49 a7 29 6d
```

Output T2:

```
e6 1a 7a 26 c4 d1 ca ee cf d8 0c ca 65 c7 1f 0f
88 c1 f8 22 c0 e8 c0 ad 94 9d 03 fe e1 39 57 9f
72 ba 0c 3d 32 c5 f9 54 f1 cc cd 54 08 1f c7 44
02 78 cb a1 fe 7b 7a 17 a9 86 fd ff 5b d1 5d 1f
```

5) PRF_IPSEC_KEYMAT_GOSTR3411_2012_256

Key K:

```
c9 a9 a7 73 20 e2 cc 55 9e d7 2d ce 6f 47 e2 19
2c ce a9 5f a6 48 67 05 82 c0 54 c0 ef 36 c2 21
```

Data of S:

```
01 26 bd b8 78 00 1d 80 60 3c 85 44 c7 27 01 00
```

Output T1:

```
21 01 d8 0c 47 db 54 bc 3c 82 9b 8c 30 7c 47 55
50 88 83 a6 d6 9e 60 1b f7 aa fb 0a bc a4 ed 95
```

Output T2:

```
33 b8 4e d0 8f 93 56 f8 1d f8 d2 79 f0 79 c9 02
87 cb 45 2c 81 d4 1e 80 38 43 08 86 c1 92 12 aa
```

6) PRF_IPSEC_PRFPLUS_GOSTR3411_2012_256

Key K:

```
c9 a9 a7 73 20 e2 cc 55 9e d7 2d ce 6f 47 e2 19
2c ce a9 5f a6 48 67 05 82 c0 54 c0 ef 36 c2 21
```

Data of S:

```
01 26 bd b8 78 00 1d 80 60 3c 85 44 c7 27 01 00
```

Output T1:

```
2d e5 ee 84 e1 3d 7b e5 36 16 67 39 13 37 0a b0
54 c0 74 b7 9b 69 a8 a8 46 82 a9 f0 4f ec d5 87
```

Output T2:

```
29 f6 0d da 45 7b f2 19 aa 2e f9 5d 7a 59 be 95
4d e0 08 f4 a5 0d 50 4d bd b6 90 be 68 06 01 53
```

7) PRF_IPSEC_KEYMAT_GOSTR3411_2012_512

Key K:

```
c9 a9 a7 73 20 e2 cc 55 9e d7 2d ce 6f 47 e2 19
2c ce a9 5f a6 48 67 05 82 c0 54 c0 ef 36 c2 21
```

Data of S:

```
01 26 bd b8 78 00 1d 80 60 3c 85 44 c7 27 01 00
```

Output T1:

```
b9 55 5b 29 91 75 4b 37 9d a6 8e 60 98 f5 b6 0e
df 91 8a 56 20 4b ff f3 a8 37 6d 1f 57 ed b2 34
a5 12 32 81 23 cd 6c 03 0b 54 14 2e 1e c7 78 2b
03 00 be a5 7c c2 a1 4c a3 b4 f0 85 a4 5c d6 ca
```

Output T2:

```
37 b1 e0 86 52 43 a4 fb 29 14 8d 27 4d 30 63 fc
bf b0 f2 f4 68 d5 27 e4 3b ca 41 fa 6b b5 3e c8
df 21 bf c4 62 3a 2e 76 8b 64 54 03 3e 09 52 32
d1 8c 86 a6 8f 00 98 d3 31 81 75 f6 59 05 ae db
```

8) PRF_IPSEC_ PRFPLUS_GOSTR3411_2012_512

Key K:

```
c9 a9 a7 73 20 e2 cc 55 9e d7 2d ce 6f 47 e2 19
2c ce a9 5f a6 48 67 05 82 c0 54 c0 ef 36 c2 21
```

Data of S:

```
01 26 bd b8 78 00 1d 80 60 3c 85 44 c7 27 01 00
```

Output T1:

```
5d a6 71 43 a5 f1 2a 6d 6e 47 42 59 6f 39 24 3f
cc 61 57 45 91 5b 32 59 10 06 ff 78 a2 08 63 d5
f8 8e 4a fc 17 fb be 70 b9 50 95 73 db 00 5e 96
26 36 98 46 cb 86 19 99 71 6c 16 5d d0 6a 15 85
```

Output T2:

```
48 34 49 5a 43 74 6c b5 3f 0a ba 3b c4 6e bc f8
77 3c a6 4a d3 43 c1 22 ee 2a 57 75 57 03 81 57
ee 9c 38 8d 96 ef 71 d5 8b e5 c1 ef a1 af a9 5e
be 83 e3 9d 00 e1 9a 5d 03 dc d6 0a 01 bc a8 e3
```

9) VKO_GOSTR3410_2012_256 with 256-bit output on the GOST R 34.10-2012 keys (512-bit output) with id-tc26-gost-3410-12-512-paramSetA

UKM value:

```
1d 80 60 3c 85 44 c7 27
```

Private key x of A:

```
c9 90 ec d9 72 fc e8 4e c4 db 02 27 78 f5 0f ca
c7 26 f4 67 08 38 4b 8d 45 83 04 96 2d 71 47 f8
c2 db 41 ce f2 2c 90 b1 02 f2 96 84 04 f9 b9 be
6d 47 c7 96 92 d8 18 26 b3 2b 8d ac a4 3c b6 67
```

Public key x*P of A (curve point (X, Y)):

```
aa b0 ed a4 ab ff 21 20 8d 18 79 9f b9 a8 55 66
54 ba 78 30 70 eb a1 0c b9 ab b2 53 ec 56 dc f5
d3 cc ba 61 92 e4 64 e6 e5 bc b6 de a1 37 79 2f
24 31 f6 c8 97 eb 1b 3c 0c c1 43 27 b1 ad c0 a7
91 46 13 a3 07 4e 36 3a ed b2 04 d3 8d 35 63 97
1b d8 75 8e 87 8c 9d b1 14 03 72 1b 48 00 2d 38
46 1f 92 47 2d 40 ea 92 f9 95 8c 0f fa 4c 93 75
64 01 b9 7f 89 fd be 0b 5e 46 e4 a4 63 1c db 5a
```

Private key y of part B:

```
48 c8 59 f7 b6 f1 15 85 88 7c c0 5e c6 ef 13 90
cf ea 73 9b 1a 18 c0 d4 66 22 93 ef 63 b7 9e 3b
80 14 07 0b 44 91 85 90 b4 b9 96 ac fe a4 ed fb
bb cc cc 8c 06 ed d8 bf 5b da 92 a5 13 92 d0 db
```

Public key y*P of B (curve point (X, Y)):

```
19 2f e1 83 b9 71 3a 07 72 53 c7 2c 87 35 de 2e
a4 2a 3d bc 66 ea 31 78 38 b6 5f a3 25 23 cd 5e
fc a9 74 ed a7 c8 63 f4 95 4d 11 47 f1 f2 b2 5c
39 5f ce 1c 12 91 75 e8 76 d1 32 e9 4e d5 a6 51
04 88 3b 41 4c 9b 59 2e c4 dc 84 82 6f 07 d0 b6
d9 00 6d da 17 6c e4 8c 39 1e 3f 97 d1 02 e0 3b
b5 98 bf 13 2a 22 8a 45 f7 20 1a ba 08 fc 52 4a
2d 77 e4 3a 36 2a b0 22 ad 40 28 f7 5b de 3b 79
```

KEK_VKO value:

```
c9 a9 a7 73 20 e2 cc 55 9e d7 2d ce 6f 47 e2 19
2c ce a9 5f a6 48 67 05 82 c0 54 c0 ef 36 c2 21
```

10) VKO_GOSTR3410_2012_512 with 512-bit output on the GOST R 34.10-2012 keys (512-bit output) with id-tc26-gost-

3410-12-512-paramSetA

UKM value:

1d 80 60 3c 85 44 c7 27

Private key x of A:

c9 90 ec d9 72 fc e8 4e c4 db 02 27 78 f5 0f ca
c7 26 f4 67 08 38 4b 8d 45 83 04 96 2d 71 47 f8
c2 db 41 ce f2 2c 90 b1 02 f2 96 84 04 f9 b9 be
6d 47 c7 96 92 d8 18 26 b3 2b 8d ac a4 3c b6 67

Public key x*P of A (curve point (X, Y)):

aa b0 ed a4 ab ff 21 20 8d 18 79 9f b9 a8 55 66
54 ba 78 30 70 eb a1 0c b9 ab b2 53 ec 56 dc f5
d3 cc ba 61 92 e4 64 e6 e5 bc b6 de a1 37 79 2f
24 31 f6 c8 97 eb 1b 3c 0c c1 43 27 b1 ad c0 a7
91 46 13 a3 07 4e 36 3a ed b2 04 d3 8d 35 63 97
1b d8 75 8e 87 8c 9d b1 14 03 72 1b 48 00 2d 38
46 1f 92 47 2d 40 ea 92 f9 95 8c 0f fa 4c 93 75
64 01 b9 7f 89 fd be 0b 5e 46 e4 a4 63 1c db 5a

Private key y of part B:

48 c8 59 f7 b6 f1 15 85 88 7c c0 5e c6 ef 13 90
cf ea 73 9b 1a 18 c0 d4 66 22 93 ef 63 b7 9e 3b
80 14 07 0b 44 91 85 90 b4 b9 96 ac fe a4 ed fb
bb cc cc 8c 06 ed d8 bf 5b da 92 a5 13 92 d0 db

Public key y*P of B (curve point (X, Y)):

19 2f e1 83 b9 71 3a 07 72 53 c7 2c 87 35 de 2e
a4 2a 3d bc 66 ea 31 78 38 b6 5f a3 25 23 cd 5e
fc a9 74 ed a7 c8 63 f4 95 4d 11 47 f1 f2 b2 5c
39 5f ce 1c 12 91 75 e8 76 d1 32 e9 4e d5 a6 51
04 88 3b 41 4c 9b 59 2e c4 dc 84 82 6f 07 d0 b6
d9 00 6d da 17 6c e4 8c 39 1e 3f 97 d1 02 e0 3b
b5 98 bf 13 2a 22 8a 45 f7 20 1a ba 08 fc 52 4a
2d 77 e4 3a 36 2a b0 22 ad 40 28 f7 5b de 3b 79

KEK_VKO value:

79 f0 02 a9 69 40 ce 7b de 32 59 a5 2e 01 52 97
ad aa d8 45 97 a0 d2 05 b5 0e 3e 17 19 f9 7b fa
7e e1 d2 66 1f a9 97 9a 5a a2 35 b5 58 a7 e6 d9
f8 8f 98 2d d6 3f c3 5a 8e c0 dd 5e 24 2d 3b df

11) Key derivation function KDF_GOSTR3411_2012_256:

K_in key:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
```

Label:

```
26 bd b8 78
```

Seed:

```
af 21 43 41 45 65 63 78
```

KDF(K_in, label, seed) value:

```
a1 aa 5f 7d e4 02 d7 b3 d3 23 f2 99 1c 8d 45 34
01 31 37 01 0a 83 75 4f d0 af 6d 7c d4 92 2e d9
```

12) Key derivation function KDF_TREE_GOSTR3411_2012_256

Output size of L:

512

K_in key:

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

Label:

26 bd b8 78

Seed:

af 21 43 41 45 65 63 78

Value of K1:

22 b6 83 78 45 c6 be f6 5e a7 16 72 b2 65 83 10
86 d3 c7 6a eb e6 da e9 1c ad 51 d8 3f 79 d1 6b

Value of K2:

07 4c 93 30 59 9d 7f 8d 71 2f ca 54 39 2f 4d dd
e9 37 51 20 6b 35 84 c8 f4 3f 9e 6d c5 15 31 f9

13) Key wrap and unwrap with the szOID_Gost28147_89_TC26_Z_ParamSet parameters

Key K:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
```

UKM value:

```
af 21 43 41 45 65 63 78
```

Label:

```
26 bd b8 78
```

KEK_e(UKM) = KDF(K_e, label, UKM):

```
a1 aa 5f 7d e4 02 d7 b3 d3 23 f2 99 1c 8d 45 34
01 31 37 01 0a 83 75 4f d0 af 6d 7c d4 92 2e d9
```

CEK_MAC:

```
38 d5 8a a3
```

CEK_ENC:

```
b9 fb 92 42 95 0f 84 3f 0f bd 5b 9a 5e cf 9f 17
f7 9e 6d 21 58 16 56 de 6d c5 85 dd 62 7a 44 0a
```

Authors' Addresses

Stanislav Smyshlyaev (editor)
CRYPTO-PRO
18, Suschevsky val
Moscow 127018
Russian Federation

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

Vladimir Popov
CRYPTO-PRO
18, Suschevsky val
Moscow 127018
Russian Federation

Email: vpopov@cryptopro.ru

Evgeny Alekseev
CRYPTO-PRO
18, Suschevsky val
Moscow 127018
Russian Federation

Email: alekseev@cryptopro.ru

Igor Oshkin
CRYPTO-PRO
18, Suschevsky val
Moscow 127018
Russian Federation

Email: oshkin@cryptopro.ru