Workgroup: Network Working Group
Internet-Draft: draft-smyslov-esp-gost-11
Published: 14 December 2021
Intended Status: Informational
Expires: 17 June 2022
Authors: V. Smyslov
ELVIS-PLUS

Using GOST ciphers in ESP and IKEv2

Abstract

This document defines a set of encryption transforms for use in the Encapsulating Security Payload (ESP) and in the Internet Key Exchange version 2 (IKEv2) protocols which are parts of the IP Security (IPsec) protocols suite. The transforms are based on the GOST R 34.12-2015 block ciphers (which are named "Magma" and "Kuznyechik") in a Multilinear Galois Mode (MGM) and the external re-keying approach.

This specification is developed to facilitate implementations that wish to support the GOST algorithms. This document does not imply IETF endorsement of the cryptographic algorithms used in this document.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The IP Security (IPsec) protocols suite consists of several protocols, of which the Encapsulating Security Payload (ESP) [RFC4303] and the Internet Key Exchange version 2 (IKEv2) [RFC7296] are most widely used. This document defines four transforms for ESP and IKEv2 based on Russian cryptographic standard algorithms (often reffered to as "GOST" algorithms). This definition is based on the Recommendations [GOST-ESP] established by Federal Agency on Technical Regulating and Metrology (Rosstandart), which describe how Russian cryptographic standard algorithms are used in ESP and IKEv2. Transforms defined in this document are based on two block ciphers from Russian cryptographic standard algorithms - "Kuznyechik" [GOST3412-2015][RFC7801] and "Magma" [GOST3412-2015][RFC8891] in Multilinear Galois Mode (MGM) [GOST-MGM][RFC9058]. These transforms provide Authenticated Encryption with Associated Data (AEAD). An external re-keying mechanism, described in [RFC8645] is also used in these transforms to limit load on session keys.

Because the GOST specification includes the definition of both 128 ("Kuznyechik") and 64 ("Magma") bit block ciphers, both are included in this document. Implementers should make themselves aware of the relative security and other cost-benefit implications of the two ciphers. See Section 5 for more details.

This specification is developed to facilitate implementations that wish to support the GOST algorithms. This document does not imply IETF endorsement of the cryptographic algorithms used in this document.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Overview

Russian cryptographic standard algorithms, often referred as "GOST" algorithms, constitute a set of cryptographic algorithms of different types - ciphers, hash functions, digital signatures, etc. In particular, Russian cryptographic standard [GOST3412-2015] defines two block ciphers - "Kuznyechik" (also defined in [RFC7801]) and "Magma" (also defined in [RFC8891]). Both ciphers use 256-bit key. "Kuznyechik" has a block size of 128 bits, while "Magma" has a 64-bit block.

Multilinear Galois Mode (MGM) is an AEAD mode defined in [GOST-MGM] [RFC9058]. It is claimed to provide defense against some attacks on well-known AEAD modes, like Galois Counter Mode (GCM).

[RFC8645] defines mechanisms that can be used to limit the number of times any particular session key is used. One of these mechanisms, called external re-keying with tree-based construction (defined in Section 5.2.3 of [RFC8645]), is used in the defined transforms. For the purpose of deriving subordinate keys a Key Derivation Function (KDF) KDF_GOSTR3411_2012_256 defined in Section 4.5 of [RFC7836], is used. This KDF is based on an HMAC [RFC2104] construction with a Russian GOST hash function defined in Russian cryptographic standard [GOST3411-2012] (also defined in [RFC6986]).

4. Transforms Description

This document defines four transforms of Type 1 (Encryption Algorithm) for use in ESP and IKEv2. All of them use MGM mode of operation with tree-based external re-keying. The transforms differ in underlying ciphers and in cryptographic services they provide.

* ENCR_KUZNYECHIK_MGM_KTREE (Transform ID 32) is an AEAD transform based on "Kuznyechik" algorithm; it provides confidentiality and message authentication and thus can be used in both ESP and IKEv2

* ENCR_MAGMA_MGM_KTREE (Transform ID 33) is an AEAD transform based on "Magma" algorithm; it provides confidentiality and message authentication and thus can be used in both ESP and IKEv2

* ENCR_KUZNYECHIK_MGM_MAC_KTREE (Transform ID 34) is a MAC-only transform based on "Kuznyechik" algorithm; it provides no confidentiality and thus can only be used in ESP, but not in IKEv2

* ENCR_MAGMA_MGM_MAC_KTREE (Transform ID 35) is a MAC-only transform based on "Magma" algorithm; it provides no confidentiality and thus can only be used in ESP, but not in IKEv2

Note that transforms ENCR_KUZNYECHIK_MGM_MAC_KTREE and ENCR_MAGMA_MGM_MAC_KTREE don't provide any confidentiality, but they are defined as Type 1 (Encryption Algorithm) transforms because of the need to include an Initialization Vector, which is impossible for Type 3 (Integrity Algorithm) transforms.

4.1. Tree-based External Re-Keying

All four transforms use the same tree-based external re-keying mechanism. The idea is that the key that is provided for the transform is not directly used to protect messages. Instead a tree of keys is derived using this key as a root. This tree may have several levels. The leaf keys are used for message protection, while intermediate nodes keys are used to derive lower level keys, including leaf keys. See Section 5.2.3 of [RFC8645] for more details. This construction allows us to protect a large amount of data, at the same time providing a bound on a number of times any particular key in the tree is used, thus defending against some side channel attacks and also increasing the key lifetime limitations based on combinatorial properties.

The transforms defined in this document use a three-level tree. The leaf key that protects a message is computed as follows:

```
K_msg = KDF (KDF (KDF (K, l1, 0x00 | i1), l2, i2), l3, i3)
```

where:

KDF (k, l, s) Key Derivation Function KDF_GOSTR3411_2012_256 defined in Section 4.5 of [RFC7836], which accepts three input parameters - a key (k), a label (l) and a seed (s) and provides a new key as an output;

K the root key for the tree (see Section 4.4);

l1, l2, l3 labels defined as 6 octet ASCII strings without null termination:

l1 = "level1"

l2 = "level2"

l3 = "level3"

i1, i2, i3 parameters that determine which keys out of the tree are used on each level, altogether they determine a leaf key that is used for message protection; the length of i1 is one octet, i2 and i3 are two octet integers in network byte order;

| indicates concatenation;

This construction allows us to generate up to 2^8 keys on level 1 and up to 2^16 keys on levels 2 and 3. So, the total number of possible leaf keys generated from a single SA key is 2^40.

This specification doesn't impose any requirements on the frequency of which the external re-keying takes place. It is expected that sending application will follow its own policy dictating how many times the keys on each level must be used.

4.2. Initialization Vector Format

Each message protected by the defined transforms MUST contain an Initialization Vector (IV). The IV has a size of 64 bits and consists of the four fields, three of which are i1, i2 and i3 parameters that determine the particular leaf key this message was protected with (see Section 4.1), and the fourth is a counter, representing the message number for this key.

```
                      1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |      i1       |               i2              |      i3       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   i3 (cont)   |                      pnum                     |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: IV Format

where:

* i1 (1 octet), i2 (2 octets), i3 (2 octets) - parameters, determining the particular key used to protect this message; 2-octets parameters are integers in network byte order

* pnum (3 octets) - message counter in network byte order for the leaf key protecting this message; up to 2^24 messages may be protected using a single leaf key

For any given SA the IV MUST NOT repeat, but there is no requirement that IV is unpredictable.

4.3. Nonce Format for MGM

MGM requires a per-message nonce (called Initial Counter Nonce, ICN, in the [RFC9058]) that MUST be unique in the context of any leaf key. The size of the ICN is n-1 bits, where n is the block size of the underlying cipher. The two ciphers used in the transforms defined in this document have different block sizes, so two different formats for the ICN are defined.

MGM specification requires that the nonce be n-1 bits in size, where n is the block size of the underlying cipher. This document defines MGM nonces having n bits (the block size of the underlying cipher) in size. Since the n is always a multiple of 8 bits, this makes MGM nonces having a whole number of octets. When used inside MGM the most significant bit of the first octet of the nonce (represented as an octet string) is dropped, making the effective size of the nonce equal to n-1 bits. Note that the dropped bit is a part of zero field (see Figure 2 and Figure 3) which is always set to 0, so no information is lost when it is dropped.

4.3.1. MGM Nonce Format for "Kuznyechik" based Transforms

For transforms based on "Kuznyechik" cipher (ENCR_KUZNYECHIK_MGM_KTREE and ENCR_KUZNYECHIK_MGM_MAC_KTREE) the ICN consists of a zero octet, a 24-bit message counter and a 96-bit secret salt, that is fixed for SA and is not transmitted.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     zero      |                      pnum                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                             salt                              |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Nonce format for "Kuznyechik" based transforms

where:

*zero (1 octet) - set to 0

*pnum (3 octets) - the counter for the messages protected by the given leaf key; this field MUST be equal to the pnum field in the IV

*salt (12 octets) - secret salt

4.3.2. MGM Nonce Format for "Magma" based Transforms

For transforms based on "Magma" cipher (ENCR_MAGMA_MGM_KTREE and ENCR_MAGMA_MGM_MAC_KTREE) the ICN consists of a zero octet, a 24-bit message counter and a 32-bit secret salt, that is fixed for SA and is not transmitted.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     zero      |                      pnum                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             salt                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Nonce format for "Magma" based transforms

where:

*zero (1 octet) - set to 0

*pnum (3 octets) - the counter for the messages protected by the given leaf key; this field MUST be equal to the pnum field in the IV

*salt (4 octets) - secret salt

4.4. Keying Material

The transform key for ENCR_KUZNYECHIK_MGM_KTREE and ENCR_KUZNYECHIK_MGM_MAC_KTREE transforms consists of 352 bits (44 octets), of which the first 256 bits is a root key for the tree (denoted as K in Section 4.1) and the remaining 96 bits is a secret salt (see Section 4.3.1).

The transform key for ENCR_MAGMA_MGM_KTREE and ENCR_MAGMA_MGM_MAC_KTREE transforms consists of 288 bits (36 octets), of which the first 256 bits is a root key for the tree (denoted as K in Section 4.1) and the remaining 32 bits is a secret salt (see Section 4.3.2).

In case of ESP the transform keys are extracted from the KEYMAT as defined in Section 2.17 of [RFC7296]. In case of IKEv2 the transform keys are either SK_ei or SK_er, which are generated as defined in Section 2.14 of [RFC7296]. Note that since these transforms provide authenticated encryption, no additional keys are needed for authentication. It means that in case of IKEv2 the keys SK_ai/SK_ar are not used and MUST be treated as having zero length.

4.5. Integrity Check Value

The length of the authentication tag that MGM can compute is in the range from 32 bits to the block size of the underlying cipher. Section 4 of the [RFC9058] states that the authentication tag length must be fixed for a particular protocol. For "Kuznyechik" based transforms (ENCR_KUZNYECHIK_MGM_KTREE and ENCR_KUZNYECHIK_MGM_MAC_KTREE) the resulting Integrity Check Value (ICV) length is set to 96 bits. For "Magma" based transforms (ENCR_MAGMA_MGM_KTREE and ENCR_MAGMA_MGM_MAC_KTREE) the full ICV length is set to the block size - 64 bits.

4.6. Plaintext Padding

Transforms defined in this document don't require any plaintext padding, as specified in [RFC9058]. It means, that only those padding requirements that are imposed by the protocol are applied (4 bytes for ESP, no padding for IKEv2).

4.7. AAD Construction

4.7.1. ESP AAD

Additional Authenticated Data (AAD) in ESP is constructed differently depending on the transform being used and whether Extended Sequence Number (ESN) is in use or not. The ENCR_KUZNYECHIK_MGM_KTREE and ENCR_MAGMA_MGM_KTREE provide confidentiality, so the content of the ESP body is encrypted and AAD consists of the ESP SPI and (E)SN. The AAD is constructed similarly to the one in [RFC4106].

On the other hand the ENCR_KUZNYECHIK_MGM_MAC_KTREE and ENCR_MAGMA_MGM_MAC_KTREE don't provide confidentiality, they provide only message authentication. For this purpose the IV and the part of ESP packet that is normally encrypted are included in the AAD. For these transforms encryption capability provided by MGM is not used. The AAD is constructed similarly to the one in [RFC4543].

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               SPI                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     32-bit Sequence Number                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: AAD for AEAD transforms with 32-bit SN

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               SPI                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 64-bit Extended Sequence Number               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: AAD for AEAD transforms with 64-bit ESN

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               SPI                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     32-bit Sequence Number                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               IV                              |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                     Payload Data (variable)                   ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Padding (0-255 bytes)                      |
+                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |  Pad Length   | Next Header   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: AAD for authentication only transforms with 32-bit SN

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               SPI                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 64-bit Extended Sequence Number               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               IV                              |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                     Payload Data (variable)                   ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Padding (0-255 bytes)                      |
+                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |  Pad Length   | Next Header   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: AAD for authentication only transforms with 64-bit ESN

4.7.2. IKEv2 AAD

For IKEv2 the AAD consists of the IKEv2 Header, any unencrypted payloads following it (if present) and the Encrypted (or the

Encrypted Fragment) payload header. The AAD is constructed similar to the one in [RFC5282].

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                         IKEv2 Header                          ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                   Unencrypted IKE Payloads                    ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8: AAD for IKEv2

4.8. Using Transforms

When SA is established the i1, i2 and i3 parameters are set to 0 by the sender and a leaf key is calculated. The pnum parameter starts from 0 and is incremented with each message protected by the same leaf key. When sender decides that the leaf should be changed, it increments i3 parameter and generates a new leaf key. The pnum parameter for the new leaf key is reset to 0 and the process continues. If the sender decides, that 3rd level key corresponding to i3 is used enough times, it increments i2, resets i3 to 0 and calculates a new leaf key. The pnum is reset to 0 (as with every new leaf key) and the process continues. Similar procedure is used when 2nd level key needs to be changed.

A combination of i1, i2, i3 and pnum MUST NOT repeat for any particular SA. This means that wrapping around of these counters is not allowed: when i2, i3 or pnum reach their maximum values, a procedure of changing a leaf key described above is executed, and if all four parameters reach their maximum values, the IPsec SA becomes unusable.

There may be other reasons to recalculate leaf keys beside reaching maximum values for the counters. For example, the sender may count a number of octets protected by a particular leaf key and generate a new key when some threshold is reached.

The receiver always uses i1, i2 and i3 from the received message. If they differ from the values in previously received packets, a new leaf key is calculated. The pnum parameter is always used from the received packet. To improve performance implementations may cache recently used leaf key. When a new leaf key is calculated (based on the values from received message) the old key may be kept for some time to improve performance in case of possible packet reordering (when packets protected by the old leaf key are delayed and arrive later).

5. Security Considerations

The most important security consideration for MGM is that the nonce MUST NOT repeat for a given key. For this reason the transforms defined in this document MUST NOT be used with manual keying.

Excessive use of the same key can give an attacker advantages in breaking security properties of the transforms defined in this document. For this reason the amound of data any particular key is used to protect should be limited. This is especially important for algorithms with 64-bit block size (like "Magma"), which nowdays are generally considered insecure after protecting relatively small amount of data. For example, Section 3.4 of [SP800-67] limits the number of blocks that are allowed to be encrypted with Triple DES cipher by 2^20 (8 Mbytes of data). This document defines a rekeying mechanism that allows to mitigate a weak security of a 64-bit block cipher by frequent changing of encryption key.

For transforms defined in this document, [GOST-ESP] recommends limiting the number of octets protected with a single Kmsg key by the following values:

* for transforms based on "Kuznyechik" cipher (ENCR_KUZNYECHIK_MGM_KTREE and ENCR_KUZNYECHIK_MGM_MAC_KTREE) - 2^41 octets;

* for transforms based on "Magma" cipher (ENCR_MAGMA_MGM_KTREE and ENCR_MAGMA_MGM_MAC_KTREE) - 2^28 octets;

These values are based on combinatorial properties and may be further restricted if side channels attacks are taken into considerations. Note that the limit for "Kuznyechik" based transforms is unreachable because due to transforms construction the number of protected messages is limited to 2^24 and each message (either IKEv2 message or ESP datagram) is limited to 2^16 octets in size, giving 2^40 octets as the maximum amount of data that can be protected with a single Kmsg.

Section 4 of [RFC9058] discusses the possibility of truncating authentication tags in MGM as a trade-off between message expansion and the forgery probability. This specification truncates an authentication tag length for "Kuznyechik" based transforms to 96 bits. This decreases message expansion still providing very low forgery probability - 2^-96.

Security properties of MGM are discussed in [MGM-SECURITY].

6. IANA Considerations

IANA maintains a registry of "Internet Key Exchange Version 2 (IKEv2) Parameters" with a sub-registry of "Transform Type Values". IANA has assigned four Transform IDs in the "Transform Type 1 - Encryption Algorithm Transform IDs" registry and is requested to update their references to this document (where RFCXXXX is this document):

Number	Name	ESP Reference	IKEv2 Reference
32	ENCR_KUZNYECHIK_MGM_KTREE	[RFCXXXX]	[RFCXXXX]
33	ENCR_MAGMA_MGM_KTREE	[RFCXXXX]	[RFCXXXX]
34	ENCR_KUZNYECHIK_MGM_MAC_KTREE	[RFCXXXX]	Not allowed
35	ENCR_MAGMA_MGM_MAC_KTREE	[RFCXXXX]	Not allowed

7. Acknowledgments

Author wants to thank Adrian Farrel, Russ Housley, Yaron Sheffer and Stanislav Smyshlyaev for valuable input in the process of publication this document.

8. References

8.1. Normative References

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/info/rfc4303>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2

(IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[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>.

[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>.

[RFC8891] Dolmatov, V., Ed. and D. Baryshkov, "GOST R 34.12-2015: Block Cipher "Magma"", RFC 8891, DOI 10.17487/RFC8891, September 2020, <https://www.rfc-editor.org/info/rfc8891>.

[RFC9058] Smyshlyaev, S., Ed., Nozdrunov, V., Shishkin, V., and E. Griboedova, "Multilinear Galois Mode (MGM)", RFC 9058, DOI 10.17487/RFC9058, June 2021, <https://www.rfc-editor.org/info/rfc9058>.

[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

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

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

[GOST-MGM] Federal Agency on Technical Regulating and Metrology, "Information technology. Cryptographic data security. Authenticated encryption block cipher operation modes", R 1323565.1.026-2019, 2019. (In Russian)

[GOST-ESP] Federal Agency on Technical Regulating and Metrology, "Information technology. Cryptographic data security. Using Russian cryptographic algorithms in data security protocol ESP", R 1323565.1.035-2021, 2021. (In Russian)

[RFC2104]
Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[RFC4106] Viega, J. and D. McGrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)", RFC 4106, DOI 10.17487/RFC4106, June 2005, <https://www.rfc-editor.org/info/rfc4106>.

[RFC4543] McGrew, D. and J. Viega, "The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH", RFC 4543, DOI 10.17487/RFC4543, May 2006, <https://www.rfc-editor.org/info/rfc4543>.

[RFC5282] Black, D. and D. McGrew, "Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol", RFC 5282, DOI 10.17487/RFC5282, August 2008, <https://www.rfc-editor.org/info/rfc5282>.

[RFC8645] Smyshlyaev, S., Ed., "Re-keying Mechanisms for Symmetric Keys", RFC 8645, DOI 10.17487/RFC8645, August 2019, <https://www.rfc-editor.org/info/rfc8645>.

[MGM-SECURITY] Akhmetzyanova, L., Alekseev, E., Karpunin, G., and V. Nozdrunov, "Security of Multilinear Galois Mode (MGM)", 2019, <https://eprint.iacr.org/2019/123.pdf>.

[SP800-67] National Institute of Standards and Technology, "Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher", November 2017, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf>.

Appendix A. Test Vectors

In the following test vectors binary data is represented in hexadecimal format. The numbers in square bracket indicate the size of the corresponding data in decimal format.

1. ENCR_KUZNYECHIK_MGM_KTREE, example 1:

```
   transform key [44]:
       b6 18 0c 14 5c 51 2d bd 69 d9 ce a9 2c ac 1b 5c
       e1 bc fa 73 79 2d 61 af 0b 44 0d 84 b5 22 cc 38
       7b 67 e6 f2 44 f9 7f 06 78 95 2e 45
   K [32]:
       b6 18 0c 14 5c 51 2d bd 69 d9 ce a9 2c ac 1b 5c
       e1 bc fa 73 79 2d 61 af 0b 44 0d 84 b5 22 cc 38
   salt [12]:
       7b 67 e6 f2 44 f9 7f 06 78 95 2e 45
   i1 = 00, i2 = 0000, i3 = 0000, pnum = 000000
   K_msg [32]:
       2f f1 c9 0e de 78 6e 06 1e 17 b3 74 d7 82 af 7b
       d8 80 bd 52 7c 66 a2 ba dc 3e 56 9a ab 27 1d a4
   nonce [16]:
       00 00 00 00 7b 67 e6 f2 44 f9 7f 06 78 95 2e 45
   IV [8]:
       00 00 00 00 00 00 00 00
   AAD [8]:
       51 46 53 6b 00 00 00 01
   plaintext [64]:
       45 00 00 3c 23 35 00 00 7f 01 ee cc 0a 6f 0a c5
       0a 6f 0a 1d 08 00 f3 5b 02 00 58 00 61 62 63 64
       65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74
       75 76 77 61 62 63 64 65 66 67 68 69 01 02 02 04
   ciphertext [64]:
       18 9d 12 88 b7 18 f9 ea be 55 4b 23 9b ee 65 96
       c6 d4 ea fd 31 64 96 ef 90 1c ac 31 60 05 aa 07
       62 97 b2 24 bf 6d 2b e3 5f d6 f6 7e 7b 9d eb 31
       85 ff e9 17 9c a9 bf 0b db af c2 3e ae 4d a5 6f
   ESP ICV [12]:
       50 b0 70 a1 5a 2b d9 73 86 89 f8 ed
   ESP packet [112]:
       45 00 00 70 00 4d 00 00 ff 32 91 4f 0a 6f 0a c5
       0a 6f 0a 1d 51 46 53 6b 00 00 00 01 00 00 00 00
       00 00 00 00 18 9d 12 88 b7 18 f9 ea be 55 4b 23
       9b ee 65 96 c6 d4 ea fd 31 64 96 ef 90 1c ac 31
       60 05 aa 07 62 97 b2 24 bf 6d 2b e3 5f d6 f6 7e
       7b 9d eb 31 85 ff e9 17 9c a9 bf 0b db af c2 3e
       ae 4d a5 6f 50 b0 70 a1 5a 2b d9 73 86 89 f8 ed
```

2. ENCR_KUZNYECHIK_MGM_KTREE, example 2:

```
   transform key [44]:
       b6 18 0c 14 5c 51 2d bd 69 d9 ce a9 2c ac 1b 5c
       e1 bc fa 73 79 2d 61 af 0b 44 0d 84 b5 22 cc 38
       7b 67 e6 f2 44 f9 7f 06 78 95 2e 45
   K [32]:
       b6 18 0c 14 5c 51 2d bd 69 d9 ce a9 2c ac 1b 5c
       e1 bc fa 73 79 2d 61 af 0b 44 0d 84 b5 22 cc 38
   salt [12]:
       7b 67 e6 f2 44 f9 7f 06 78 95 2e 45
   i1 = 00, i2 = 0001, i3 = 0001, pnum = 000000
   K_msg [32]:
       9a ba c6 57 78 18 0e 6f 2a f6 1f b8 d5 71 62 36
       66 c2 f5 13 0d 54 e2 11 6c 7d 53 0e 6e 7d 48 bc
   nonce [16]:
       00 00 00 00 7b 67 e6 f2 44 f9 7f 06 78 95 2e 45
   IV [8]:
       00 00 01 00 01 00 00 00
   AAD [8]:
       51 46 53 6b 00 00 00 10
   plaintext [64]:
       45 00 00 3c 23 48 00 00 7f 01 ee b9 0a 6f 0a c5
       0a 6f 0a 1d 08 00 e4 5b 02 00 67 00 61 62 63 64
       65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74
       75 76 77 61 62 63 64 65 66 67 68 69 01 02 02 04
   ciphertext [64]:
       78 0a 2c 62 62 32 15 7b fe 01 76 32 f3 2d b4 d0
       a4 fa 61 2f 66 c2 bf 79 d5 e2 14 9b ac 1d fc 4b
       15 4b 69 03 4d c2 1d ef 20 90 6d 59 62 81 12 7c
       ff 72 56 ab f0 0b a1 22 bb 5e 6c 71 a4 d4 9a 4d
   ESP ICV [12]:
       c2 2f 87 40 83 8e 3d fa ce 91 cc b8
   ESP packet [112]:
       45 00 00 70 00 5c 00 00 ff 32 91 40 0a 6f 0a c5
       0a 6f 0a 1d 51 46 53 6b 00 00 00 10 00 00 01 00
       01 00 00 00 78 0a 2c 62 62 32 15 7b fe 01 76 32
       f3 2d b4 d0 a4 fa 61 2f 66 c2 bf 79 d5 e2 14 9b
       ac 1d fc 4b 15 4b 69 03 4d c2 1d ef 20 90 6d 59
       62 81 12 7c ff 72 56 ab f0 0b a1 22 bb 5e 6c 71
       a4 d4 9a 4d c2 2f 87 40 83 8e 3d fa ce 91 cc b8
```

3. ENCR_MAGMA_MGM_KTREE, example 1:

```
   transform key [36]:
       5b 50 bf 33 78 87 02 38 f3 ca 74 0f d1 24 ba 6c
       22 83 ef 58 9b e6 f4 6a 89 4a a3 5d 5f 06 b2 03
       cf 36 63 12
   K [32]:
       5b 50 bf 33 78 87 02 38 f3 ca 74 0f d1 24 ba 6c
       22 83 ef 58 9b e6 f4 6a 89 4a a3 5d 5f 06 b2 03
   salt [4]:
       cf 36 63 12
   i1 = 00, i2 = 0000, i3 = 0000, pnum = 000000
   K_msg [32]:
       25 65 21 e2 70 b7 4a 16 4d fc 26 e6 bf 0c ca 76
       5e 9d 41 02 7d 4b 7b 19 76 2b 1c c9 01 dc de 7f
   nonce [8]:
       00 00 00 00 cf 36 63 12
   IV [8]:
       00 00 00 00 00 00 00 00
   AAD [8]:
       c8 c2 b2 8d 00 00 00 01
   plaintext [64]:
       45 00 00 3c 24 2d 00 00 7f 01 ed d4 0a 6f 0a c5
       0a 6f 0a 1d 08 00 de 5b 02 00 6d 00 61 62 63 64
       65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74
       75 76 77 61 62 63 64 65 66 67 68 69 01 02 02 04
   ciphertext [64]:
       fa 08 40 33 2c 4f 3f c9 64 4d 8c 2c 4a 91 7e 0c
       d8 6f 8e 61 04 03 87 64 6b b9 df bd 91 50 3f 4a
       f5 d2 42 69 49 d3 5a 22 9e 1e 0e fc 99 ac ee 9e
       32 43 e2 3b a4 d1 1e 84 5c 91 a7 19 15 52 cc e8
   ESP ICV [8]:
       5f 4a fa 8b 02 94 0f 5c
   ESP packet [108]:
       45 00 00 6c 00 62 00 00 ff 32 91 3e 0a 6f 0a c5
       0a 6f 0a 1d c8 c2 b2 8d 00 00 00 01 00 00 00 00
       00 00 00 00 fa 08 40 33 2c 4f 3f c9 64 4d 8c 2c
       4a 91 7e 0c d8 6f 8e 61 04 03 87 64 6b b9 df bd
       91 50 3f 4a f5 d2 42 69 49 d3 5a 22 9e 1e 0e fc
       99 ac ee 9e 32 43 e2 3b a4 d1 1e 84 5c 91 a7 19
       15 52 cc e8 5f 4a fa 8b 02 94 0f 5c
```

4. ENCR_MAGMA_MGM_KTREE, example 2:

```
   transform key [36]:
       5b 50 bf 33 78 87 02 38 f3 ca 74 0f d1 24 ba 6c
       22 83 ef 58 9b e6 f4 6a 89 4a a3 5d 5f 06 b2 03
       cf 36 63 12
   K [32]:
       5b 50 bf 33 78 87 02 38 f3 ca 74 0f d1 24 ba 6c
       22 83 ef 58 9b e6 f4 6a 89 4a a3 5d 5f 06 b2 03
   salt [4]:
       cf 36 63 12
   i1 = 00, i2 = 0001, i3 = 0001, pnum = 000000
   K_msg [32]:
       20 e0 46 d4 09 83 9b 23 f0 66 a5 0a 7a 06 5b 4a
       39 24 4f 0e 29 ef 1e 6f 2e 5d 2e 13 55 f5 da 08
   nonce [8]:
       00 00 00 00 cf 36 63 12
   IV [8]:
       00 00 01 00 01 00 00 00
   AAD [8]:
       c8 c2 b2 8d 00 00 00 10
   plaintext [64]:
       45 00 00 3c 24 40 00 00 7f 01 ed c1 0a 6f 0a c5
       0a 6f 0a 1d 08 00 cf 5b 02 00 7c 00 61 62 63 64
       65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74
       75 76 77 61 62 63 64 65 66 67 68 69 01 02 02 04
   ciphertext [64]:
       7a 71 48 41 a5 34 b7 58 93 6a 8e ab 26 91 40 a8
       25 a7 f3 5d b9 e4 37 1f e7 6c 99 9c 9b 88 db 72
       1d c7 59 f6 56 b5 b3 ea b6 b1 4d 6b d7 7a 07 1d
       4b 93 78 bd 08 97 6c 33 ed 9a 01 91 bf fe a1 dd
   ESP ICV [8]:
       dd 5d 50 9a fd b8 09 98
   ESP packet [108]:
       45 00 00 6c 00 71 00 00 ff 32 91 2f 0a 6f 0a c5
       0a 6f 0a 1d c8 c2 b2 8d 00 00 00 10 00 00 01 00
       01 00 00 00 7a 71 48 41 a5 34 b7 58 93 6a 8e ab
       26 91 40 a8 25 a7 f3 5d b9 e4 37 1f e7 6c 99 9c
       9b 88 db 72 1d c7 59 f6 56 b5 b3 ea b6 b1 4d 6b
       d7 7a 07 1d 4b 93 78 bd 08 97 6c 33 ed 9a 01 91
       bf fe a1 dd dd 5d 50 9a fd b8 09 98
```

5. ENCR_KUZNYECHIK_MGM_MAC_KTREE, example 1:

```
   transform key [44]:
       98 bd 34 ce 3b e1 9a 34 65 e4 87 c0 06 48 83 f4
       88 cc 23 92 63 dc 32 04 91 9b 64 3f e7 57 b2 be
       6c 51 cb ac 93 c4 5b ea 99 62 79 1d
   K [32]:
       98 bd 34 ce 3b e1 9a 34 65 e4 87 c0 06 48 83 f4
       88 cc 23 92 63 dc 32 04 91 9b 64 3f e7 57 b2 be
   salt [12]:
       6c 51 cb ac 93 c4 5b ea 99 62 79 1d
   i1 = 00, i2 = 0000, i3 = 0000, pnum = 000000
   K_msg [32]:
       98 f1 03 01 81 0a 04 1c da dd e1 bd 85 a0 8f 21
       8b ac b5 7e 00 35 e2 22 c8 31 e3 e4 f0 a2 0c 8f
   nonce [16]:
       00 00 00 00 6c 51 cb ac 93 c4 5b ea 99 62 79 1d
   IV [8]:
       00 00 00 00 00 00 00 00
   AAD [80]:
       3d ac 92 6a 00 00 00 01 00 00 00 00 00 00 00 00
       45 00 00 3c 0c f1 00 00 7f 01 05 11 0a 6f 0a c5
       0a 6f 0a 1d 08 00 48 5c 02 00 03 00 61 62 63 64
       65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74
       75 76 77 61 62 63 64 65 66 67 68 69 01 02 02 04
   plaintext [0]:
   ciphertext [0]:
   ESP ICV [12]:
       ca c5 8c e5 e8 8b 4b f3 2d 6c f0 4d
   ESP packet [112]:
       45 00 00 70 00 01 00 00 ff 32 91 9b 0a 6f 0a c5
       0a 6f 0a 1d 3d ac 92 6a 00 00 00 01 00 00 00 00
       00 00 00 00 45 00 00 3c 0c f1 00 00 7f 01 05 11
       0a 6f 0a c5 0a 6f 0a 1d 08 00 48 5c 02 00 03 00
       61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70
       71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
       01 02 02 04 ca c5 8c e5 e8 8b 4b f3 2d 6c f0 4d
```

6. ENCR_KUZNYECHIK_MGM_MAC_KTREE, example 2:

```
   transform key [44]:
       98 bd 34 ce 3b e1 9a 34 65 e4 87 c0 06 48 83 f4
       88 cc 23 92 63 dc 32 04 91 9b 64 3f e7 57 b2 be
       6c 51 cb ac 93 c4 5b ea 99 62 79 1d
   K [32]:
       98 bd 34 ce 3b e1 9a 34 65 e4 87 c0 06 48 83 f4
       88 cc 23 92 63 dc 32 04 91 9b 64 3f e7 57 b2 be
   salt [12]:
       6c 51 cb ac 93 c4 5b ea 99 62 79 1d
   i1 = 00, i2 = 0000, i3 = 0001, pnum = 000000
   K_msg [32]:
       02 c5 41 87 7c c6 23 f3 f1 35 91 9a 75 13 b6 f8
       a8 a1 8c b2 63 99 86 2f 50 81 4f 52 91 01 67 84
   nonce [16]:
       00 00 00 00 6c 51 cb ac 93 c4 5b ea 99 62 79 1d
   IV [8]:
       00 00 00 00 01 00 00 00
   AAD [80]:
       3d ac 92 6a 00 00 00 06 00 00 00 00 01 00 00 00
       45 00 00 3c 0c fb 00 00 7f 01 05 07 0a 6f 0a c5
       0a 6f 0a 1d 08 00 43 5c 02 00 08 00 61 62 63 64
       65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74
       75 76 77 61 62 63 64 65 66 67 68 69 01 02 02 04
   plaintext [0]:
   ciphertext [0]:
   ESP ICV [12]:
       ba bc 67 ec 72 a8 c3 1a 89 b4 0e 91
   ESP packet [112]:
       45 00 00 70 00 06 00 00 ff 32 91 96 0a 6f 0a c5
       0a 6f 0a 1d 3d ac 92 6a 00 00 00 06 00 00 00 00
       01 00 00 00 45 00 00 3c 0c fb 00 00 7f 01 05 07
       0a 6f 0a c5 0a 6f 0a 1d 08 00 43 5c 02 00 08 00
       61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70
       71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
       01 02 02 04 ba bc 67 ec 72 a8 c3 1a 89 b4 0e 91
```

7. ENCR_MAGMA_MGM_MAC_KTREE, example 1:

```
   transform key [36]:
       d0 65 b5 30 fa 20 b8 24 c7 57 0c 1d 86 2a e3 39
       2c 1c 07 6d fa da 69 75 74 4a 07 a8 85 7d bd 30
       88 79 8f 29
   K [32]:
       d0 65 b5 30 fa 20 b8 24 c7 57 0c 1d 86 2a e3 39
       2c 1c 07 6d fa da 69 75 74 4a 07 a8 85 7d bd 30
   salt [4]:
       88 79 8f 29
   i1 = 00, i2 = 0000, i3 = 0000, pnum = 000000
   K_msg [32]:
       4c 61 45 99 a0 a0 67 f1 94 87 24 0a e1 00 e1 b7
       ea f2 3e da f8 7e 38 73 50 86 1c 68 3b a4 04 46
   nonce [8]:
       00 00 00 00 88 79 8f 29
   IV [8]:
       00 00 00 00 00 00 00 00
   AAD [80]:
       3e 40 69 9c 00 00 00 01 00 00 00 00 00 00 00 00
       45 00 00 3c 0e 08 00 00 7f 01 03 fa 0a 6f 0a c5
       0a 6f 0a 1d 08 00 36 5c 02 00 15 00 61 62 63 64
       65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74
       75 76 77 61 62 63 64 65 66 67 68 69 01 02 02 04
   plaintext [0]:
   ciphertext [0]:
   ESP ICV [8]:
       4d d4 25 8a 25 35 95 df
   ESP packet [108]:
       45 00 00 6c 00 13 00 00 ff 32 91 8d 0a 6f 0a c5
       0a 6f 0a 1d 3e 40 69 9c 00 00 00 01 00 00 00 00
       00 00 00 00 45 00 00 3c 0e 08 00 00 7f 01 03 fa
       0a 6f 0a c5 0a 6f 0a 1d 08 00 36 5c 02 00 15 00
       61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70
       71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
       01 02 02 04 4d d4 25 8a 25 35 95 df
```

8. ENCR_MAGMA_MGM_MAC_KTREE, example 2:

```
        transform key [36]:
            d0 65 b5 30 fa 20 b8 24 c7 57 0c 1d 86 2a e3 39
            2c 1c 07 6d fa da 69 75 74 4a 07 a8 85 7d bd 30
            88 79 8f 29
        K [32]:
            d0 65 b5 30 fa 20 b8 24 c7 57 0c 1d 86 2a e3 39
            2c 1c 07 6d fa da 69 75 74 4a 07 a8 85 7d bd 30
        salt [4]:
            88 79 8f 29
        i1 = 00, i2 = 0000, i3 = 0001, pnum = 000000
        K_msg [32]:
            b4 f3 f9 0d c4 87 fa b8 c4 af d0 eb 45 49 f2 f0
            e4 36 32 b6 79 19 37 2e 1e 96 09 ea f0 b8 e2 28
        nonce [8]:
            00 00 00 00 88 79 8f 29
        IV [8]:
            00 00 00 00 01 00 00 00
        AAD [80]:
            3e 40 69 9c 00 00 00 06 00 00 00 00 01 00 00 00
            45 00 00 3c 0e 13 00 00 7f 01 03 ef 0a 6f 0a c5
            0a 6f 0a 1d 08 00 31 5c 02 00 1a 00 61 62 63 64
            65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74
            75 76 77 61 62 63 64 65 66 67 68 69 01 02 02 04
        plaintext [0]:
        ciphertext [0]:
        ESP ICV [8]:
            84 84 a9 23 30 a0 b1 96
        ESP packet [108]:
            45 00 00 6c 00 18 00 00 ff 32 91 88 0a 6f 0a c5
            0a 6f 0a 1d 3e 40 69 9c 00 00 00 06 00 00 00 00
            01 00 00 00 45 00 00 3c 0e 13 00 00 7f 01 03 ef
            0a 6f 0a c5 0a 6f 0a 1d 08 00 31 5c 02 00 1a 00
            61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70
            71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
            01 02 02 04 84 84 a9 23 30 a0 b1 96
```

Author's Address

Valery Smyslov
ELVIS-PLUS
PO Box 81
Moscow (Zelenograd)
124460
Russian Federation

Phone: +7 495 276 0211
Email: svan@elvis.ru