1. We defined a profile of TLS 1.2 with GOST algorithms similarly to RFC 8998. Therefore, we changed the first paragraph in the Introduction to the following:

This document specifies three new cipher suites, two new signature algorithms, seven new supported groups and two new certificate types for the Transport Layer Security (TLS) Protocol Version 1.2 [[RFC5246]](file:///C:\Users\nikiforova\Downloads\tls_1_2%20(7).html#RFC5246) to support the set of Russian cryptographic standard algorithms (called GOST algorithms). This document specifies a profile of TLS 1.2 with GOST algorithms so that implementers can produce interoperable implementations. The profile of TLS 1.2 with GOST algorithms uses the hash algorithm GOST R 34.11-2012 [[RFC6986]](file:///C:\Users\nikiforova\Downloads\tls_1_2%20(7).html#RFC6986) and the signature algorithm GOST R 34.10-2012 [[RFC7091]](file:///C:\Users\nikiforova\Downloads\tls_1_2%20(7).html#RFC7091) and use two types of cipher suites: the CTR\_OMAC cipher suites and the CNT\_IMIT cipher suite.

In addition, we changed the last two paragraphs in the Introduction to the following:

This document specifies the profile of the TLS protocol version 1.2 with GOST algorithms. The profile of the TLS protocol version 1.3 [[RFC8446]](file:///D:\svn\analytics\trunk\%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F\IETF\TLS\TLS%201.2\tls_1_2.html#RFC8446) with GOST algorithms is specified in a separate document [[DraftGostTLS13]](file:///D:\svn\analytics\trunk\%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F\IETF\TLS\TLS%201.2\tls_1_2.html#DraftGostTLS13).

This specification is developed to facilitate implementations that wish to support the GOST algorithms. This document does not imply IETF endorsement of the cipher suites, signature algorithms, supported groups and certificate types.

We similarly changed the Abstract:

This document specifies three new cipher suites, two new signature algorithms, seven new supported groups and two new certificate types for the Transport Layer Security (TLS) protocol Version 1.2 to support the Russian cryptographic standard algorithms (called GOST algorithms). This document specifies a profile of TLS 1.2 with GOST algorithms so that implementers can produce interoperable implementations.

This specification is developed to facilitate implementations that wish to support the GOST algorithms. This document does not imply IETF endorsement of the cipher suites, signature algorithms, supported groups and certificate types.

In Section 4.2 we changed

All of the cipher suites described in this document use a key encapsulation mechanism based on Diffie-Hellman to share the TLS premaster secret.

to:

The profile of TLS 1.2 with GOST algorithms described in this document uses a key encapsulation mechanism based on Diffie-Hellman to share the TLS premaster secret.

2. Comment:

«The cipher suite definition in Section 4.1 attempts to place restrictions on what TLS compression method can be used with these ciphers ("MUST be 'null'" is the phrase in question). While TLS compression is now considered to be a bad idea, was removed in TLS 1.3, etc., TLS 1.2 itself does not admit the possibility for a cipher suite to restrict what compression methods it can be used with. Attempting to make the non-use of TLS compression a requirement of the cipher suite breaks a protocol abstraction of TLS. (It would not be a problem, however, to note that the same authorities that require the use of a GOST cipher suite also require the non-use of TLS compression, as a non-normative statement.) The actual protocol mechanisms described in the document seem to be compatible with the generic TLS abstraction, since references are made to TLSCompressed as opposed to TLSPlaintext.

Similarly, the discussion of the CertificateRequest handshake message in Section 4.2.3 attempts to limit the values sent for supported\_signature\_algorithms and certificate\_types when these cipher suites are used, which is attempting to redefine core TLS protocol semantics.»

Reply:

In Section 4.1 we changed:

The compression method (see [Section 6.2.2 of [RFC5246]](https://datatracker.ietf.org/doc/html/rfc5246#section-6.2.2)) MUST be

"null" in all of the cipher suites described in this document. This

compression method is negotiated according to [Section 4.2.1](https://datatracker.ietf.org/doc/html/draft-smyshlyaev-tls12-gost-suites-13#section-4.2.1). Note

that the CompressionMethod.null operation is an identity operation;

no fields of the TLSCompressed structure are altered.

to:

The profile of TLS 1.2 with GOST algorithms requires that the compression is not used.

In Section 4.2 we changed:

The cipher suites specified in this document define the ClientHello, ServerHello, server Certificate, CertificateRequest, ClientKeyExchange, CertificateVerify and Finished handshake messages, that are described in further detail below.

to:

This section specifies the data structures and computations used by the profile of TLS 1.2 with GOST algorithms. The specifications for the ClientHello, ServerHello, server Certificate, CertificateRequest, ClientKeyExchange, CertificateVerify and Finished handshake messages are described in further detail below.

3. Comment:

«What is to happen to the IANA ciphersuite registrations for ciphers that are allocated and list this draft as their reference, but do not appear in this draft?:

0xC1,0x03 TLS\_GOSTR341112\_256\_WITH\_KUZNYECHIK\_MGM\_L N N

[draft-smyshlyaev-tls13-gost-suites] 0xC1,0x04

TLS\_GOSTR341112\_256\_WITH\_MAGMA\_MGM\_L N N

[draft-smyshlyaev-tls13-gost-suites] 0xC1,0x05

TLS\_GOSTR341112\_256\_WITH\_KUZNYECHIK\_MGM\_S N N

[draft-smyshlyaev-tls13-gost-suites] 0xC1,0x06

TLS\_GOSTR341112\_256\_WITH\_MAGMA\_MGM\_S N N

[draft-smyshlyaev-tls13-gost-suites]»

Reply:

These IANA ciphersuite registrations are used in TLS protocol version 1.3. They are defined in <https://datatracker.ietf.org/doc/html/draft-smyshlyaev-tls13-gost-suites-04>

4. Comment:

«Section 3

i & j bitwise AND of integers i and j;

bitwise AND of negative integers requires knowledge of the representation convention (e.g., twos-complement). Perhaps there is a presumption that all integers in question are unsigned?

P\_s the point of order q\_s that belongs to the same curve as Q\_s;

Is there really only a single such point? My understanding is that nearly all elements of the (sub)group generated by P\_s would have the same order, for prime-order groups. In such a case, I would guess that referring to P\_s as the distinguished/well-known generator would be more accurate.»

Reply:

We changed the definitions to the following:

i & j

bitwise AND of unsigned integers i and j;

P\_s

the distinguished generator of the subgroup of order q\_s that belongs to the same curve as Q\_s;

5. Comment:

«Section 4.1

All of the cipher suites described in this document use the stream

cipher (see Section 4.3.3) to protect records. The TLSCiphertext

structure for the CTR\_OMAC and CNT\_IMIT cipher suites is specified in

accordance with the Standard Stream Cipher case (see Section 6.2.3.1

of [RFC5246]):

While the use of the indicated RFC 5246 Standard Stream Cipher abstraction is appropriate for many block cipher modes of operation (such as the counter-based ones employed here), the referenced Section

4.3.3 does not seem to actually describe how the cipher mode of operation is translated into functioning as a stream cipher, so as to justify this statement.»

Reply:

We changed

All of the cipher suites described in this document use the stream cipher (see [Section 4.3.3](https://datatracker.ietf.org/doc/html/draft-smyshlyaev-tls12-gost-suites-05#section-4.3.3)) to protect records.

to:

All of the cipher suites described in this document use such modes of operation (see [Section 4.3.3](https://datatracker.ietf.org/doc/html/draft-smyshlyaev-tls12-gost-suites-05#section-4.3.3)) that protect the records in the same way as if they were protected by a stream cipher.

6. Comment:

«Section 4.1.1, 4.1.2

I strongly recommend some explicit discussion of the interaction between these ciphers and the encrypt-then-mac extension. While the current formulation claims to use the RFC 5246 GenericStreamCipher representation (for which RFC 7366 says encrypt-then-mac should not be negotiated), the actual operation does not actually reflect the GenericStreamCipher format, since the MACValue\_seqnum is covered by the encryption operation. (It also redefines the MAC() operation to use a per-record key, so the RFC 7366 expressions are not directly transferrable, either.) [ed. I see that there is some mention of encrypt-then-mac down in §4.2.1. It might be useful here as well.]»

Reply:

We added the following paragraph at the end of Section 4.1.1:

Note that the profile of TLS 1.2 with GOST algorithms uses the authenticate-then-encrypt method (see Appendix F.4 of [[RFC5246]](file:///D:\svn\analytics\trunk\%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F\IETF\TLS\TLS%201.2\tls_1_2.html#RFC5246)). The profile of TLS 1.2 with GOST algorithms requires that the encrypt\_then\_mac extension is not used in the ServerHello message (see [Section 4.2.1](file:///D:\svn\analytics\trunk\%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F\IETF\TLS\TLS%201.2\tls_1_2.html#Hello)).

We added the following paragraph at the end of Section 4.1.2:

Note that the profile of TLS 1.2 with GOST algorithms uses the authenticate-then-encrypt method (see Appendix F.4 of [[RFC5246]](file:///D:\svn\analytics\trunk\%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F\IETF\TLS\TLS%201.2\tls_1_2.html#RFC5246)). The profile of TLS 1.2 with GOST algorithms requires that the encrypt\_then\_mac extension is not used in the ServerHello message (see [Section 4.2.1](file:///D:\svn\analytics\trunk\%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F\IETF\TLS\TLS%201.2\tls_1_2.html#Hello)).

7. Comment:

«Section 4.2

The cipher suites specified in this document define the ClientHello,

ServerHello, server Certificate, CertificateRequest,

ClientKeyExchange, CertificateVerify and Finished handshake messages,

that are described in further detail below.

At risk of excessive pedanticism, the cipher suite cannot "define" the hello messages, since those are used to negotiate what cipher suite is to be used. The ClientHello message format is a protocol invariant, and the ServerHello message format is defined by the protocol version in use (invariant to what cipher suite is used).»

Reply:

We changed this text to the following:

This section specifies the data structures and computations used by the profile of TLS 1.2 with GOST algorithms. The specifications for the ClientHello, ServerHello, server Certificate, CertificateRequest, ClientKeyExchange, CertificateVerify and Finished handshake messages are described in further detail below.

8. Comment:

«Section 4.2.1

o The ClientHello.compression\_methods field MUST contain exactly one

byte, set to zero, which corresponds to the "null" compression

method.

(As mentioned above, this is not in the remit for a cipher suites to

mandate.)».

Reply:

We described the Handshake messages for the profile of TLS 1.2 with GOST algorithms (see point 7). Consequently, we do not change the semantics of the protocol but only define the order of the protocol with GOST algorithms.

9. Comment:

«o The ClientHello.extensions field MUST contain the

signature\_algorithms extension (see [RFC5246]).

If the negotiated cipher suite is one of CTR\_OMAC/CTR\_IMIT and the

client implementation does not support generating the

signature\_algorithms extension with the values defined in

Section 5, the server MUST either abort the connection or ignore

this extension and behave as if the client had sent the

signature\_algorithms extension with the values {8, 64} and {8,

65}.

This setup seems a little confused. First off, the first "MUST" isn't really a MUST, since you go on to specify that a handshake can succeed without such an extension. (It's also rather unusual and surprising for a cipher suite to require an extension to be included if the cipher suite is advertised.) Second, the server has no way of knowing that the client implementation "does not support" generating the signature\_algorithms extension, just that the client ended up not generating it this time around. Saying that the server has to either abort or use a default if the server negotiates the cipher and the extension is missing is okay, though (but it would be even better to remove the choice and say always abort or always assume the default).»

Reply:

There are two points:

1) It may not be possible to add a value to the ClientHello signature\_algorithms extension in some implementations, for example, in Microsoft implementation.

2) According to RFC 5246 the client «supports only the default hash and signature algorithms» (defined in RFC 5246) if the ClientHello.extensions field does not contain the signature\_algorithms extension.

Because of the first point, we use an alternative variant if signature\_algorithms extension does not contain the required values. And because of the second point, the ClientHello.extensions field MUST contain the signature\_algorithms extension.

We added some changes in the text:

* The ClientHello.extensions field MUST contain the signature\_algorithms extension (see [[RFC5246]](file:///C:\Users\nikiforova\Downloads\tls_1_2%20(3).html#RFC5246)).  
    
  If the negotiated cipher suite is one of CTR\_OMAC/CTR\_IMIT and the **signature\_algorithms extension in the ClientHello message does not contain the values** defined in [Section 5](file:///C:\Users\nikiforova\Downloads\tls_1_2%20(3).html#SAR), the server MUST either abort the connection or ignore this extension and behave as if the client had sent the signature\_algorithms extension with the values {8, 64} and {8, 65}.

10. Comment:

« o The ServerHello.compression\_method field MUST contain exactly one

byte, set to zero, which corresponds to the "null" compression

method.

(same as above)»

Reply:

We described the Handshake messages for the profile of TLS 1.2 with GOST algorithms (see point 7). Consequently, we do not change the semantics of the protocol but only define the order of the protocol with GOST algorithms.

11. Comment:

«Section 4.2.3

o the CertificateRequest.supported\_signature\_algorithm field MUST

contain only signature/hash algorithm pairs with the values {8,

64} or {0, 65} defined in Section 5;

Is the "{0,65}" a typo for "{8,65}"? I'm not sure why the 256-bit version of gostr34102012 would map up to the "intrinsic" hash but the 512-bit version use the "none" hash.

(Also, per the high-level comment, we really need to be treating these pairs as coming from the TLS SignatureScheme registry, and not the separate HashAlgorithm and SignatureAlgorithm registries.)»

Reply:

Thanks a lot! It is a typo.

12. Comment:

«Section 4.2.4.1

3. Generates an export representation PSExp of the premaster secret

PS using the KExp15 algorithm defined in Section 8.2.1:

(The steps listed here don't actually show the client computing the PS value itself, just PSExp.)»

Reply:

We added the following step:

2. Generates the premaster secret PS, where PS is chosen from B\_32 at random.

13. Comment:

« 4. Generates the ClientKeyExchange message using the

GostKeyTransport structure that is defined as follows:

GostKeyTransport ::= SEQUENCE {

keyExp OCTET STRING,

ephemeralPublicKey SubjectPublicKeyInfo,

ukm OCTET STRING OPTIONAL

Do we care what ASN.1 encoding rules are used to encode the GostKeyTransport structure for conveyance in the ClientKeyExchange?

(It's a little weird to use the same name for the TLS presentation-language object and the ASN.1 object whose encoding it contains; typically we would have the TLS presentation-language object be defined as an opaque[length] and use the prose to specify what it contains (and, in this case, that its length is determined by the containing Handshake structure).

where the keyExp field contains the PSExp value, the

ephemeralPublicKey field contains the Q\_eph value and the ukm field

MUST be ignored by the server.

We would often say what the ukm contains (zero-length octet string?) in addition to that the recipient ignores its contents.»

Reply:

The DER encoding rules is used to encode the GostKeyTransport structure.

We added the following sentence at the end of Section 4.2.4:

The DER encoding rules are used to encode the GostKeyTransport and the TLSGostKeyTransportBlob structures.

The ukm value from GostKeyTransport is not used. The document does not fixed its contents. So, the server ignores its contents.

14. Comment:

«Section 4.2.4.2

In case of the CNT\_IMIT cipher suite the body of the

ClientKeyExchange message consists of a TLSGostKeyTransportBlob

structure that is defined bellow.

We probably want to say something about what ASN.1 encoding rules are used.»

Reply:

As said above, we added the following sentence at the end of Section 4.2.4:

The DER encoding rules are used to encode the GostKeyTransport and the TLSGostKeyTransportBlob structures.

15. Comment:

« 3. Generates an export representation PSExp of the premaster secret

PS using the KExp28147 algorithm defined in Section 8.2.2:

As above, the actual value of PS used by the client does not seem to be specified.»

Reply:

We added the following step:

2. Generates the premaster secret PS, where PS is chosen from B\_32 at random.

16. Comment:

«Section 4.3.2

The CNT\_IMIT cipher suite uses the message authentication code

function gostIMIT28147 defined in Section 8.4 with the initialization

vector IV = IV0, where IV0 in B\_8 is a string of all zeros, with the

CryptoPro Key Meshing algorithm defined in [RFC4357]. The resulting

MAC length is 4 bytes and the MAC key length is 32 bytes.

A 32-bit MAC seems short enough to merit particular mention, whether here or in the security considerations section.»

Reply:

The CNT\_IMIT is an old cipher suite. However, it is still widely used in Russia. Because of this, we describe it in this document.

We added the following paragraph in Security Considerations:

The authenticate-then-encrypt method is crucial for the CNT\_IMIT cipher suite. Encryption of the MAC value is conducted to reduce the possibility of forgery to guessing. Here the probability of guess is approximately equal to 2^{-32}, which is acceptable in some practical cases.

17. Comment:

«Section 4.3.3

As alluded to by an earlier comment, I suggest mentioning here that the use of a counter mode makes the cipher effectively function as a stream cipher, so the TLS GenericStreamCipher case is (almost) applicable.

(The "almost" refers to the MAC being encrypted. In theory one could make the encryption part of the definition of the MAC operation and have the formalism line up, but that seems like it would be very painful to write up accurately, and is almost certainly not worth the effort. At most I would add some hedging language up where we talk about GenericStreamCipher to clarify that though it's technically not correct in practice it is fine.)»

Reply:

We added the following paragraph at the end of Section 4.3.3:

Note that the counter modes used in cipher suites described in this document act as stream ciphers.

19. Comment:

«Section 8.2.1

The keys K\_Exp\_MAC and K\_Exp\_ENC MUST be independent. For every pair

of keys (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.

Hmm, it seems that we use an IV extracted from H(c\_r|s\_r), which is not exactly preshared or sent with the export key representation»

Reply:

The Section 8.2.1 describes the KExp15 and KImp15 algorithms separately from the TLS protocol. A «preshared value» here is the value that is known (out of band) before using the KImp15 algorithm. Thus, in TLS 1.2 protocol the IV is preshared value for the KImp15 algorithm.

We changed the text to the following:

The keys K\_Exp\_MAC and K\_Exp\_ENC MUST be independent. For every pair of keys (K\_Exp\_ENC, K\_Exp\_MAC) the IV values MUST be unique. For the import of key with the KImp15 algorithm, the IV value may be sent with the export key representation.

20. Comment:

«Section 10

Note that prior to the existence of this document implementations

could use only the values from the Private Use space in order to use

the GOST-based algorithms. So some old implementations can still use

the old value {0x00, 0x81} instead of the {0xC1, 0x02} value to

indicate the TLS\_GOSTR341112\_256\_WITH\_28147\_CNT\_IMIT cipher suite;

The "old value" {0x00, 0x81} is not in the private-use range for cipher suites; was {0xff, 0x81} intended?»

Reply:

Thanks a lot. It is a typo. The correct old value is {0xFF, 0x85}.

21. Comment:

«Client should be prepared to handle any of them correctly if

corresponding group is included in the supported\_groups extension

(see [RFC8422] and [RFC7919]).

I'm not entirely sure that I'm interpreting this correctly -- the GC256[BCD] groups are defined in Table 2 and have IANA-allocated codepoints. Is this saying that if I receive one of those group codepoints, I might receive points that actually correspond to the groups listed in Table 8, as opposed to the "real" groups that are listed in table 2? I don't see these parameter set (OID names) listed in either of RFCs 7836 or 4357; are they available somewhere that can be referenced? (Even a Russian-language reference seems preferable to no reference at all, in this case.)»

Reply:

The Table 8 is used only to notice that there are another handles for the groups from Table 2 that can be found in the Internet. This table does not contain any new groups.

We added the references in text in following way:

Due to historical reasons in addition to the curve identifier values listed in Table 2 there exist some extra identifier values that correspond to the curves GC256B, GC256C and GC256D as follows (see [[RFC4357]](https://xml2rfc.tools.ietf.org/cgi-bin/cat.cgi/tls_1_2.html?input=94fedff8d188fd182ec0976525d802f729fdc211691d-1629183065#RFC4357), [[R-1323565.1.024-2019]](https://xml2rfc.tools.ietf.org/cgi-bin/cat.cgi/tls_1_2.html?input=94fedff8d188fd182ec0976525d802f729fdc211691d-1629183065#R-1323565.1.024-2019)).

22. Comment:

«Section 11

Typically we would note here that the use of a static DH key/cert by the server means that these ciphers do not provide forward secrecy.»

Reply:

We added the following sentence in Security Considerations:

The profile of TLS 1.2 with GOST algorithms does not provide Perfect Forward Secrecy.

23. About IANA

We have the allocated values for the TLS protocol version 1.3. They correspond to the gostr34102012\_256 and gostr34102012\_512 algorithms respectively but with deterministic elliptic curves. We think it is a good idea to make the values 0x0840 and 0x0841 «Reserved for backward compatibility».

Similarly to the RFC 8422, we added the following paragraph at the end of Section 5:

So, to represent gostr34102012\_256 and gostr34102012\_512 in the signature\_algorithms extension, the value shall be (8,64) and (8,65), respectively.