The Security Evaluated Standardized Password Authenticated Key Exchange (SESPAKE) Protocol
draft-smyshlyaev-sespake-16

Abstract

This document specifies the Security Evaluated Standardized Password Authenticated Key Exchange (SESPAKE) protocol. The SESPAKE protocol provides password authenticated key exchange for usage in the systems for protection of sensitive information. The security proofs of the protocol were made for the case of an active adversary in the channel, including MitM attacks and attacks based on the impersonation of one of the subjects.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 26, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
Table of Contents

1. Introduction
2. Conventions Used in This Document
3. Notations
4. Protocol Description
   4.1. Protocol Parameters
   4.2. Initial Values of the Protocol Counters
   4.3. Protocol Steps
5. Construction of Points Q_1,...,Q_N
6. Acknowledgments
7. Security Considerations
8. References
   8.1. Normative References
   8.2. Informative References
Appendix A. Test Examples for GOST-based Protocol Implementation
   A.1. Examples of Points
   A.2. Test Examples of SESPAKE
Appendix B. Point Verification Script
Authors' Addresses

1. Introduction

The current document contains the description of the password authenticated key exchange protocol SESPAKE (security evaluated standardized password authenticated key exchange) for usage in the systems for protection of sensitive information. The protocol is intended to use for establishment of keys that are then used for organization of secure channel for protection of sensitive information. The security proofs of the protocol were made for the case of an active adversary in the channel, including MitM attacks and attacks based on the impersonation of one of the subjects.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Notations

This document uses the following parameters of elliptic curves in accordance with [RFC6090]:

\[ E \]
\[ p \]
\[ a, b \]
\[ m \]
\[ q \]
\[ P \]

E — an elliptic curve defined over a finite prime field GF(p), where \( p > 3 \);

p — the characteristic of the underlying prime field;

a, b — the coefficients of the equation of the elliptic curve in the canonical form;

m — the elliptic curve group order;

q — the elliptic curve subgroup order;

P — a generator of the subgroup of order q;
X, Y
the coordinates of the elliptic curve point in the canonical form;
O
zero point (point of infinity) of the elliptic curve.

This memo uses the following functions:

HASH
the underlying hash function;

HMAC
the function for calculating a message authentication code, based on a HASH function in accordance with [RFC2104];

F(PW, salt, n)
the value of the function PBKDF2(PW,salt,n,len), where PBKDF2(PW,salt,n,len) is calculated according to [RFC2898] The parameter len is considered equal to minimal integer that is a multiple of 8 and satisfies the following condition:
len >= floor(log_2(q)).

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

B_n
the set of byte strings of size n, n >= 0, for n = 0 the B_n set consists of a single empty string of size 0; if b is an element of B_n, then b = (b_1,...,b_n), where b_1,...,b_n are elements of {0,…,255};

||
concatenation of byte strings A and C, i.e., if A in B_n1, C in B_n2, A = (a_1,a_2,...,a_n1) and C = (c_1,c_2,...,c_n2), then A||C = (a_1,a_2,...,a_n1,c_1,c_2,...,c_n2) is an element of B_(n1+n2);

int(A)
for the byte string A= (a_1,...,a_n) in B_n an integer int(A) = 256^(n-1)a_n +...+ 256^(0)a_1;

bytes_n(X)
the byte string A in B_n such that int(A) = X, where X is integer, 0 <= X < 256^n;

BYTES(Q)
for Q in E, the byte string bytes_n(X) || bytes_n(Y), where X, Y are standard Weierstrass coordinates of point Q and n = ceil(log_{256}(p)).

4. Protocol Description

The main point of the SESPAKE protocol is that parties sharing a weak key (a password) generate a strong common key. The active adversary who has an access to a channel is not able to obtain any information that can be used to find a key in offline mode, i.e. without interaction with legitimate participants.

The protocol is used by the subjects A (client) and B (server) that share some secret parameter that was established in an out-of-band mechanism: a client is a participant who stores a password as a secret parameter and a server is a participant who stores a password-based computed point of the elliptic curve.

The SESPAKE protocol consists of two steps: the key agreement step and the key confirmation step. During the first step (the key agreement step) the parties exchange keys using Diffie-Hellman with public components masked by an element that depends on the password - one of the predefined elliptic curve points multiplied by the password-based coefficient. This approach provides an implicit key authentication, which means that after this step one party is assured that no other party aside from a specifically identified second party may gain access to the generated secret key. During the second step (the key confirmation step) the parties exchange strings that strongly depend on the generated key. After this step the parties are assured that a legitimate party and no one else actually has possession of the secret key.
To protect against online guessing attacks the failed connections counters were introduced in the SESPAKE protocol. There is also a special way of a small order point processing and a mechanism that provides a reflection attack protection by using different operations for different sides.

### 4.1. Protocol Parameters

Various elliptic curves can be used in the protocol. For each elliptic curve supported by clients the following values MUST be defined:

- the protocol parameters identifier ID_ALG (which can also define a HASH function, PRF used in PBKDF2 function, etc.), that is a byte string of an arbitrary length;
- the point \( P \), that is a generator point of the subgroup of order \( q \) of the curve;
- the set of distinct curve points \( \{Q_1, Q_2, \ldots, Q_N\} \) of order \( q \), where the total number of points \( N \) is defined for protocol instance.

The method of generation of the points \( \{P, Q_1, Q_2, \ldots, Q_N\} \) is described in Section 5.

The protocol parameters that are used by subject A are the following:

1. The secret password value \( PW \), which is a byte string that is uniformly randomly chosen from a subset of cardinality \( 10^{10} \) or greater of the set \( B_k \), where \( k \geq 6 \) is password length.
2. The list of curve identifiers supported by A.
3. Sets of points \( \{Q_1, Q_2, \ldots, Q_N\} \), corresponding to curves supported by A.
4. The C\(_1\)_A counter, that tracks the total number of unsuccessful authentication trials in a row, and a value of CLim\(_1\) that stores the maximum possible number of such events.
5. The C\(_2\)_A counter, that tracks the total number of unsuccessful authentication events during the period of usage of the specific PW, and a value of CLim\(_2\) that stores the maximum possible number of such events.
6. The C\(_3\)_A counter, that tracks the total number of authentication events (successful and unsuccessful) during the period of usage of the specific PW, and a value of CLim\(_3\) that stores the maximum possible number of such events.
7. The unique identifier ID\(_A\) of the subject A (OPTIONAL), which is a byte string of an arbitrary length.

The protocol parameters that are used by subject B are the following:

1. The values ind and salt, where ind is in \([1, \ldots, N]\), salt is in \([1, \ldots, 2^{128}]-1\).
2. The point \( Q_{PW} \), satisfying the following equation:

\[
Q_{PW} = \text{int} \left( F (PW, \text{salt}, 2000) \right) \cdot Q_{\text{ind}}.
\]

It is possible that the point \( Q_{PW} \) is not stored and is calculated using \( PW \) in the beginning of the protocol. In that case B has to store \( PW \) and points \( Q_1, Q_2, \ldots, Q_N \).

3. The ID\(_B\) identifier.
4. The C\(_1\)_B counter, that tracks the total number of unsuccessful authentication trials in a row, and a value of CLim\(_1\) that stores the maximum possible number of such events.
5. The C\(_2\)_B counter, that tracks the total number of unsuccessful authentication events during the period of usage of the specific PW, and a value of CLim\(_2\) that stores the maximum possible number of such events.
6. The C\(_3\)_B counter, that tracks the total number of authentication events (successful and unsuccessful) during the period of usage of the specific PW, and a value of CLim\(_3\) that stores the maximum possible number of such events.
7. The unique identifier ID\(_B\) of the subject B (OPTIONAL), which is a byte string of an arbitrary length.
### 4.2. Initial Values of the Protocol Counters

After the setup of a new password value PW the values of the counters MUST be assigned as follows:

- \( C_1^A = C_1^B = CLim_1 \), where \( CLim_1 \) is in \( \{3, \ldots, 5\} \);
- \( C_2^A = C_2^B = CLim_2 \), where \( CLim_2 \) is in \( \{7, \ldots, 20\} \);
- \( C_3^A = C_3^B = CLim_3 \), where \( CLim_3 \) is in \( \{10^3, 10^3+1, \ldots, 10^5\} \).

### 4.3. Protocol Steps

The basic SESPAKE steps are shown in the scheme below:

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>if ( C_1^A ) or ( C_2^A ) or ( C_3^A ) = 0 ==&gt; quit</td>
<td>if ( C_1^B ) or ( C_2^B ) or ( C_3^B ) = 0 ==&gt; quit</td>
</tr>
<tr>
<td>decrement ( C_1^A, C_2^A, C_3^A ) by 1</td>
<td>decrement ( C_1^B, C_2^B, C_3^B ) by 1</td>
</tr>
<tr>
<td>( z_A = 0 )</td>
<td>( z_B = 0 )</td>
</tr>
<tr>
<td>Q_PW^A = int(F(PW, salt, 2000)) * Q_ind</td>
<td>Q_B = ( u_2 + Q_PW )</td>
</tr>
<tr>
<td>choose alpha randomly from ( {1, \ldots, q-1} )</td>
<td>choose beta randomly from ( {1, \ldots, q-1} )</td>
</tr>
<tr>
<td>( u_1 = alpha\cdot P - Q_PW^A )</td>
<td>if ( m/q*Q_B = O ) ==&gt; ( Q_B = beta\cdot P, z_B = 1 )</td>
</tr>
<tr>
<td>( z_B = 0 )</td>
<td>( K_B = HASH(BYTES((m/q*beta(mod q))*Q_B)) )</td>
</tr>
<tr>
<td>( Q_A = u_2 - Q_PW^A )</td>
<td>if ( u_2 ) not in ( E ) ==&gt; quit</td>
</tr>
<tr>
<td>if ( m/q*Q_A = O ) ==&gt; ( Q_A = alpha\cdot P, z_A = 1 )</td>
<td>( u_2 = betta\cdot P + Q_PW )</td>
</tr>
<tr>
<td>( K_A = HASH(BYTES((m/q*alpha(mod q))*Q_A)) )</td>
<td></td>
</tr>
<tr>
<td>( U_1 = BYTES(u_1), U_2 = BYTES(u_2) )</td>
<td></td>
</tr>
<tr>
<td>MAC_A = HMAC(K_A, 0x01</td>
<td></td>
</tr>
<tr>
<td>U_1 = MAC_A, U_2 = MAC_B</td>
<td>U_1 = MAC_A, U_2 = MAC_B</td>
</tr>
<tr>
<td>DATA_A, MAC_A ---&gt;</td>
<td>DATA_A, MAC_B ---&gt;</td>
</tr>
<tr>
<td>if MAC_A != HMAC(K_B, 0x01</td>
<td></td>
</tr>
<tr>
<td>if ( z_A = 1 ) == quit</td>
<td>if ( z_B = 1 ) == quit</td>
</tr>
<tr>
<td>if MAC_B != HMAC(K_B, 0x02</td>
<td></td>
</tr>
</tbody>
</table>

\( C_1^B = CLim_1 \), increment \( C_2^B \) by 1
The full description of the protocol consists of the following steps:

1. If any of the counters $C_1^A$, $C_2^A$, $C_3^A$ is equal to 0, A finishes the protocol with an error that informs of exceeding the number of trials that is controlled by the corresponding counter.
2. A decrements each of the counters $C_1^A$, $C_2^A$, $C_3^A$ by 1, requests open authentication information from B and sends the ID_A identifier.
3. If any of the counters $C_1^B$, $C_2^B$, $C_3^B$ is equal to 0, B finishes the protocol with an error that informs of exceeding the number of trials that is controlled by the corresponding counter.
4. B decrements each of the counters $C_1^B$, $C_2^B$, $C_3^B$ by 1.
5. B sends the values of ind, salt and the ID_ALG identifier to A. B also can OPTIONALLY send the ID_B identifier to A. All following calculations are done by B in the elliptic curve group defined by the ID_ALG identifier.
6. A sets the curve defined by the received ID_ALG identifier as the used elliptic curve. All following calculations are done by A in this elliptic curve group.
7. A calculates the point $Q_{PW}^A = \text{int}(F(PW, salt, 2000)) \cdot Q_{ind}$.
8. A chooses randomly (according to the uniform distribution) the value alpha, alpha is in {1, ..., q-1}, and assigns $z_A = 0$.
9. A sends the value $u_1 = \alpha \cdot P - Q_{PW}^A$ to B.
10. After receiving $u_1$, B checks that $u_1$ is in E. If it is not, B finishes with an error, considering the authentication process unsuccessful.
11. B calculates $Q_B = u_1 + Q_{PW}$, assigns $z_B = 0$ and chooses randomly (according to the uniform distribution) the value betta, betta is in {1, ..., q-1}.
12. If $m/q \cdot Q_B = O$, B assigns $Q_B = \beta \cdot P$ and $z_B = 1$.
13. B calculates $K_B = \text{HASH} \left( \text{BYTES} \left( \frac{m}{q} \cdot \beta \cdot (\text{mod } q) \cdot Q_B \right) \right)$.
14. B sends the value $u_2 = \beta \cdot P + Q_{PW}$ to A.
15. After receiving $u_2$, A checks that $u_2$ is in E. If it is not, A finishes with an error, considering the authentication process unsuccessful.
16. A calculates $Q_A = u_2 - Q_{PW}^A$.
17. If $m/q \cdot Q_A = O$, then A assigns $Q_A = \alpha \cdot P$ and $z_A = 1$.
18. A calculates $K_A = \text{HASH} \left( \text{BYTES} \left( \frac{m}{q} \cdot \alpha \cdot (\text{mod } q) \cdot Q_A \right) \right)$.
19. A calculates $U_1 = \text{BYTES}(u_1)$, $U_2 = \text{BYTES}(u_2)$.
20. A calculates $MAC_A = \text{HMAC}(K_A, 0x01 || ID_A || ind || salt || U_1 || U_2 || ID_ALG \text{ (OPTIONAL)} || DATA_A)$, where $DATA_A$ is an OPTIONAL string that is authenticated with MAC_A (if it is not used, then $DATA_A$ is considered to be of zero length).
21. A sends $DATA_A$, MAC_A to B.
22. B calculates $U_1 = \text{BYTES}(u_1)$, $U_2 = \text{BYTES}(u_2)$.
23. B checks that the values $MAC_A$ and $\text{HMAC}(K_B, 0x01 || ID_A || ind || salt || U_1 || U_2 || ID_ALG \text{ (OPTIONAL)} || DATA_A)$ are equal. If they are not, it finishes with an error, considering the authentication process unsuccessful.
24. If $z_B = 1$, B finishes, considering the authentication process unsuccessful.
25. B sets the value of $C_1^B$ to $CLim_1$ and increments $C_2^B$ by 1.
26. B calculates $MAC_B = \text{HMAC}(K_B, 0x02 || ID_B || ind || salt || U_1 || U_2 || ID_ALG \text{ (OPTIONAL)} || DATA_A || DATA_B)$, where $DATA_B$ is an OPTIONAL string that is authenticated with MAC_B (if it is not used, then $DATA_B$ is considered to be of zero length).
27. B sends $DATA_B$, MAC_B to A.
28. A checks that the values $MAC_B$ and $\text{HMAC}(K_A, 0x02 || ID_B || ind || salt || U_1 || U_2 ||
ID_ALG (OPTIONAL) || DATA_A || DATA_B) are equal. If they are not, it finishes with an error, considering the authentication process unsuccessful.

29. If z_A = 1, A finishes, considering the authentication process unsuccessful.

30. A sets the value of C_1^A to CLim_1 and increments C_2^A by 1.

After the successful finish of the procedure the subjects A and B are mutually authenticated and each subject has an explicitly authenticated value of K = K_A = K_B.

Notes:

1. In the case when the interaction process can be initiated by any subject (client or server) the ID_A and ID_B options MUST be used and the receiver MUST check that the identifier he had received is not equal to his own, otherwise, it finishes the protocol. If an OPTIONAL parameter ID_A (or ID_B) is not used in the protocol, it SHOULD be considered equal to a fixed byte string (zero-length string is allowed) defined by a specific implementation.

2. The ind, ID_A, ID_B and salt parameters can be agreed in advance. If some parameter is agreed in advance, it is possible not to send it during a corresponding step. Nevertheless, all parameters MUST be used as corresponding inputs to HMAC function during stages 20, 23, 26 and 28.

3. The ID_ALG parameter can be fixed or agreed in advance.

4. The ID_ALG parameter is RECOMMENDED to be used in HMAC during stages 20, 23, 26 and 28.

5. Continuation of protocol interaction in case of any of the counters C_1^nA, C_1^nB being equal to zero MAY be done without changing password. In this case these counters can be used for protection against denial-of-service attacks. For example, continuation of interaction can be allowed after a certain delay.

6. Continuation of protocol interaction in case of any of the counters C_2^nA, C_3^nA, C_2^nB, C_3^nB being equal to zero MUST be done only after changing password.

7. It is RECOMMENDED that during the stages 9 and 14 the points u_1 and u_2 are sent in a non-compressed format (BYTES(u_1) and BYTES(u_2)). However, the point compression MAY be used.

8. The use of several Q points can reinforce the independence of the data streams in case of working with several applications, when, for example, two high-level protocols can use two different points. However, the use of more than one point is OPTIONAL.

5. Construction of Points Q_1,...,Q_N

This section provides an example of possible algorithm for generation of each point Q_i in the set \{Q_1,...,Q_N\} that corresponds to the given elliptic curve E.

The algorithm is based on choosing points with coordinates with known preimages of a cryptographic hash function H, which is the GOST R 34.11-2012 hash function (see [RFC6986]) with 256-bit output, if $2^{254} < q < 2^{256}$, and the GOST R 34.11-2012 hash function (see [RFC6986]) with 512-bit output, if $2^{508} < q < 2^{512}$.

The algorithm consists of the following steps:

1. Set i = 1, SEED = 0, s = 4.
2. Calculate $X = \text{int}((\text{HASH}(\text{BYTES}(P)) || \text{bytes}_s(\text{SEED})) \mod p$.
3. Check that the value of $X^3 + aX + b$ is a quadratic residue in the field F_p. If it is not, set SEED = SEED + 1 and return to Step 2.
4. Choose the value of $Y = \min(r_1, r_2)$, where $r_1, r_2$ from \{0,1,...,p-1\} are such that $r_1 \neq r_2$ and $r_1^2 + r_2 = R \mod p$ for $R = X^3 + aX + b$.
5. Check that for the point $Q = (X,Y)$ the following relations hold: $Q \neq O$ and $q*Q = O$. If they do, go to Step 6; if not, set SEED = SEED + 1 and return to Step 2.
6. Set \( Q_i = Q \). If \( i < N \), then set \( i = i+1 \) and go to Step 2, else finish.

With the defined algorithm for any elliptic curve E point sets \( \{Q_1, \ldots, Q_N\} \) are constructed. Constructed points in one set MUST have distinct X-coordinates.

Note: The knowledge of a hash function preimage prevents knowledge of the multiplicity of any point related to generator point \( P \). It is of primary importance, because such a knowledge could be used to implement an attack against protocol with exhaustive search of password.

6. Acknowledgments

We thank Lolita Sonina, Georgiy Borodin, Sergey Agafin and Ekaterina Smyshlyaeva for their careful readings and useful comments.

7. Security Considerations

Any cryptographic algorithms, particularly HASH function and HMAC function, that are used in the SESPAKE protocol MUST be carefully designed and MUST be able to withstand all known types of cryptanalytic attack.

It is RECOMMENDED that the HASH function satisfies the following condition:
\[
\text{hashlen} \leq \log_2(q) + 4,
\]
where hashlen is the lengths of the HASH function output.

The output length of hash functions that are used in the SESPAKE protocol is RECOMMENDED to be greater or equal to 256 bits.

The points \( Q_1, Q_2, \ldots, Q_N \) and \( P \) MUST be chosen in such a way that they are provable pseudorandom. As a practical matter, this means that the algorithm for generation of each point \( Q_i \) in the set \( \{Q_1, \ldots, Q_N\} \) (see Section 5) ensures that multiplicity of any point under any other point is unknown.

For a certain ID_ALG using \( N = 1 \) is RECOMMENDED.

Note: The exact adversary models, which have been considered during the security evaluation, can be found in the paper [SESPAKE-SECURITY], containing the security proofs.

8. References

8.1. Normative References


8.2. Informative References


Appendix A. Test Examples for GOST-based Protocol Implementation

The following test examples are made for the protocol implementation that is based on the Russian national standards GOST R 34.10-2012 [GOST3410-2012] and GOST R 34.11-2012 [GOST3411-2012]. The English versions of these standards can be found in [RFC7091] and [RFC6986].

A.1. Examples of Points

There is one point Q_1 for each of the elliptic curves below. These points were constructed using the method described in Section 5, in case when N = 1, where the GOST R 34.11-2012 hash function (see [RFC6986]) with 256-bit output is used if 2^254 < q < 2^256, the GOST R 34.11-2012 hash function (see [RFC6986]) with 512-bit output is used if 2^508 < q < 2^512.

Each of the points complies with the GOST R 34.10-2012 [GOST3410-2012] standard and is represented by a pair of (X, Y) coordinates in the canonical form and by a pair of (U, V) coordinates in the twisted Edwards form in accordance with the document [RFC7836] for the curves that have the equivalent representations in this form. There is a SEED value for each point, by which it was generated.

Point Q_1
X = 0xa69d51caf1a309fa9e9b66187759b0174c274e080356f23cfcbe84d396ad7bb
Y = 0x5d26f29ecc2e9ac0404df7986fa55fe94986362170f54b9616426a59786dac
SEED = 0x0001

A.1.2. Curve id-GostR3410-2001-CryptoPro-B-ParamSet
Point Q_1
X = 0x3d715a874a4b17cb3b5178793a9794a2b36c89d2f6c693f01e4cc27e7f49e399
Y = 0x1c5a641fcf7ce7e87c6f8ceea38f3db3096eace2fad158384b5395365f4fe7fe
SEED = 0x0000

A.1.3. Curve id-GostR3410-2001-CryptoPro-C-ParamSet
Point Q_1
X = 0x1e36383e43bb6cfa2917167d71b7b5dd3d6d462b43d7c64282ae67dfbec2559d
Y = 0x137478a9f721c73932ea6b45cf7e37eb78a63f29a542e563c614650c8b6399
SEED = 0x0006

A.1.4. Curve id-tc26-gost-3410-2012-512-paramSetA
Point Q_1
X = 0x2a17f833a32795327478871b5c5e88aefb91126c64b4b8327289bea62559425
d1819f133f400874328b220c74497cd240586cb249e158532cb8090776cd61c
Y = 0x728f0c4a73b48da41ce928358fad26b47a6e094e9362bae82559f83cdc4ec3a
4676bd3707edae4fd85e99695c64241edc622be87dc0cf87f51f4367f723c5
A.1.5. Curve id-tc26-gost-3410-2012-512-paramSetB
Point Q_1
X = 0x7e1fae8285e035bec244bef2d0e5ebf436633cf50e55231dea9c9cf21d4c8c33
df85d4305de92971f0a4b4c07e00d87bdcb720eb66649079285aaf12e0171149
Y = 0x2cc9998b875d446380508d858a196592db20a161558f2f4e7a85725d209
53967ae821afdeae89bb77c83a2528ef6fece02f68bda4679d7f2704947dbc408
SEED = 0x0001

A.1.6. Curve id-tc26-gost-3410-2012-256-paramSetA
Point Q_1
X = 0xb51adf93a40ab15792164fad33352f95b66369eb2a4ef5e9ae3282932036350e
Y = 0x74a358cc08593612f5955d249c96af7e8b0bb6d8bd2bbe491046650d822be18
U = 0x2ebe97affed0f8878b80b11486de430ac2b34564e4420a2472987305bc48aeaa
V = 0x828f2dcf8f06612b4f4ea4da72ca509c0f76dd37df424ea22bfa6f4f65748c1e4
SEED = 0x0001

A.1.7. Curve id-tc26-gost-3410-2012-512-paramSetC
Point Q_1
X = 0x489c91784e02e98f19a803aba319917f37689e5a18965251ce2fffe8d8b298f
5ba7470f9e0713487f964a8397b3d09a270c9d367eb5e0656aadeeb5158f1d
Y = 0x684ea885ac064a1f3b3e3c6082a3b3e3b011db6f18e203ff87028d6eb5db
2c144f0dccc71276542bf72ca2a43f4f4939da669a60793cc704a8c9416f18
U = 0x3a3496f97e96b3849a4fa7db60fd93858bde89958e4beebd05a6b3214216b37c
9d9a560076e7ea5971828b18f8f1996f1c98b3dc9f2d3c0ed36a0d6ace8
V = 0x52d884c8bf0ad6c5f7b3973e32a668daa11fd392ff138dae6203b2ccdec561
4746d35fecz4b72725d0eb143074712c76550c7a54ff3ea26f70059480dcb50
SEED = 0x0013

A.2. Test Examples of SESPAKE

This protocol implementation uses the GOST R 34.11-2012 hash function (see [RFC6986]) with 256-bit output as the H function and the HMAC_GOSTR3411_2012_512 function defined in [RFC7836] as a PRF function for the F function. The parameter len is considered equal to 256, if $2^{254} < q < 2^{256}$, and equal to 512, if $2^{508} < q < 2^{512}$. The test examples for the point of each curve in Appendix A.1 are given below.

A.2.1 Curve id-GostR3410-2001-CryptoPro-A-ParamSet
The input protocol parameters in this example take the following values:
N = 1
ind = 1
ID_A:
00 00 00 00
ID_B:
00 00 00 00
PW:
31 32 33 34 35 36 (123456)
salt:
29 23 BE 84 E1 6C D6 AE 52 90 49 F1 F1 BB E9 EB
Q_ind:
X = 0xA69D51CAF1A9B6E618759B0174C274E080356F23CFCBFE84D396AD7BB
Y = 0xD526F29ECC2E9AC0404DCF7986FA55FE94986362170F54B9616426A659786DAC
The function $F(PW,\text{salt},2000)$ takes the following values:

$$F(PW,\text{salt},2000):$$

BD 04 67 3F 71 49 B1 8E 98 15 5B D1 E2 72 4E 71
D0 09 9A A2 51 74 F7 92 D3 32 6C 6F 18 12 70 67

The coordinates of the point $Q_{PW}$ are:

$$X = \text{0x}59495655D1E7C7424C622485F575CCF121F3122D274101E8AB734CC9C9A9B45E$$
$$Y = \text{0x}48D1C311D3C9B701F3B03618562A4A07A044E3AF31E3999E67B487778B53C62$$

During the calculation of the message $u_1$ on the subject $A$ the parameter alpha, the point $\alpha^*P$ and the message $u_1$ take the following values:

$$\alpha = \text{0xF25380975A031FA68BBB4C384D12BDE47B7061C0D5E24993E0C873CDBA6B3}$$

$$\alpha^*P:$$

$$X = \text{0xBB}C77CF42DC1E62D06227935379B4AA4D14FEA4F565DDF4CB4FA4D31579F9676$$
$$Y = \text{0x}E16604A4AFDF28246684D4996274781F6C80ABBBA1414C1513EC988509DABF$$

$$u_1:$$

$$X = \text{0x}204F564383B2A7601A3D84FDB7E947B929BE9363DF5$$
$$Y = \text{0xE84F9E442C61DDE37B601A7F37E7C411C56183FA071DFA9320EDE7521F9D41}$$

During processing a message $u_1$, calculation the $K_B$ key and the message $u_2$ on the subject $B$ the parameters betta, src, $K_B$ = HASH(src), betta*P and $u_2$ take the following values:

$$\beta = \text{0xDC497D9EF6324912FDC678400E509A2032AE71CA0890D133B45F596FCCBD45D}$$

$$\text{src:}$$

$$2E 01 A3 D8 4F DB 7E 94 7B B8 92 9B E9 36 3D F5$$
$$F7 25 D6 40 1A A5 59 D4 1A 67 24 F8 D5 F1 8E 2C$$
$$A0 DB A9 31 05 CD DA F4 BF AE A3 90 6F DD 71 9D$$
$$BE B2 97 B6 A1 7F 4F BD 96 DC C7 23 EA 34 72 A9$$

$$K_B:$$

$$1A 62 65 54 92 1D C2 E9 2B 4D D8 D6 7D BE 5A 56$$
$$62 E5 62 99 37 3F 06 79 95 35 AD 26 09 4E CA A3$$

$$\beta^*P:$$

$$X = \text{0x}6097341C1BE388E3E7CA2DF47FAB8662271FD942E5B72EB2409E49F742BC9$$
$$Y = \text{0xC81AA48BDB4CA6FA0EF18B9788E25FE308577A6B3992217F96ED151BAB7D0}$$

$$u_2:$$

$$X = \text{0xDC137A2F1D4A35AEB0ECBF6D3486DF8480BFDC752A86D4F207D71910E22D}$$
$$Y = \text{0x}7532F0CE99DCC772A4D77861DAE57C138F07AE304A727907FBOA9DB624ED572$$

During processing a message $u_2$ and calculation the $K_A$ key the message MAC_A=HMAC ($K_A$, 0x01 || ID_A || ind || salt || $u_1$ || $u_2$) from the subject $A$ takes the following value:

$$K_A:$$

$$1A 62 65 54 92 1D C2 E9 2B 4D D8 D6 7D BE 5A 56$$
$$62 E5 62 99 37 3F 06 79 95 35 AD 26 09 4E CA A3$$

The message MAC_A=HMAC ($K_A$, 0x01 || ID_A || ind || salt || $u_1$ || $u_2$) from the subject $A$ takes the following value:

$$\text{MAC}_A:$$

$$23 7A 03 C3 5F 49 17 CE 86 B3 58 94 45 F1 1E 1A$$
$$6F 10 8B 2F DD 0A A9 E8 10 66 4B 25 59 60 B5 79$$

The message MAC_B=HMAC ($K_B$, 0x02 || ID_B || ind || salt || $u_1$ || $u_2$) from the subject $B$ takes the following value:

$$\text{MAC}_B:$$

$$9E E0 E8 73 3B 06 98 50 80 4D 97 98 73 1D CD 1C$$
$$FF E8 7A 3B 15 1F 0A E8 3E A9 6A FB 4F FC 31 E4$$

A.2.2 Curve id-GostR3410-2001-CryptoPro-B-ParamSet

The input protocol parameters in this example take the following values:

$N = 1$
ind = 1
ID_A:
00 00 00 00
ID_B:
00 00 00 00
PW:
31 32 33 34 35 36 ('123456')
salt:
29 23 BE 84 E1 6C D6 AE 52 90 49 F1 F1 BB E9 EB
Q_ind:
X = 0x3D715A874A4B17CB3B517893A9794A2B36C89D2FFC693F01EE4CC27E7F49E399
Y = 0x1C5A641FCF7E87CD8FCEA38F3DB3096EACE2FAD158384B53953365F4FE7FE
The function F (PW, salt, 2000) takes the following values:
F(PW,salt,2000):
BD 04 67 3F 71 49 B1 8E 98 15 5B D1 E2 72 4E 71
D0 09 9A A2 51 74 F7 92 D3 32 6C 6F 18 12 70 67
The coordinates of the point Q_PW are:
X = 0x6DC2AE26BC691FCA5A73D9C452790D15E34BA5404D92955B9148C2662ABB895
Y = 0x3B02AA9DD65AE30C335CED12F3154BBAC059F66B088306747453EDF6E5DB077
During the calculation of the message u_1 on the subject A the parameter
alpha, the point alpha*P and the message u_1 take the following values:
alpha=0x499D72B90299CAB0DA1F8BE19D9122F622A13B32B730C46BD0664044F2144FAD
alpha*:P
X = 0x61D6F916DB717222D74877F179F7EBEF7CD4D248C1F523C048E34A1DF308DD
Y = 0x3EC48863049CFCFE662904082E78503F4973A4E105E2F1B18C69A5E7FB209000
u_1:
X = 0x21F5437AF332D21A7111A0702264AE82D3765CD00EBFF1ECEFE1588EC50C63AB1
Y = 0x5C9553B5D11AAECE738AD9A9F8CB43C100AD4FAE089D3CBCCEA8C0172EB7ECC
During processing a message u_1, calculation the K_B key and the message
u_2 on the subject B the parameters betta, src, K_B = HASH(src), betta*P
and u_2 take the following values:
betta=0xF69FF614957EF83668EDC2D7ED614BE76F7B253DB23C5CC9C52BF7DF8F4669D
src:
50 14 0A 5D ED 33 43 EF C8 25 7B 79 E6 46 D9 F0
DF 43 82 8C 04 91 9B D4 60 C9 7A D1 4B A3 A8 6B
00 C4 06 B5 74 4D 8E B1 49 DC 8E 7F C8 40 64 D8
53 20 25 3E 57 A9 B6 B1 3D 0D 38 FE A8 EE 5E 0A
K_B:
A6 26 DE 01 B1 68 0F F7 51 30 09 12 2B CE E1 89
68 83 39 4F 96 03 01 72 45 5C 9A E0 60 CC E4 4A
betta*P:
X = 0x33BC6F7E9C0B311C0FB2B72546C32717295508EA97F8C8BA9F890F2478AB4D6C
Y = 0x75D57B396C396F492F057E9222CCC686437A2AAD464E452EF426FC8EEED1A4A6
u_2:
X = 0x89DDEE718EE8A224A7F37E22CFD731C25FCBF58860364EE322412CDCEEF99AC
Y = 0x0ECE03D4E395A6354C571871BEF425A532D5463B0F8FD427F91A43E20CDA55C
During processing a message u_2 and calculation the key on the subject A
the K_A key takes the following value:
K_A:
A6 26 DE 01 B1 68 0F F7 51 30 09 12 2B CE E1 89
68 83 39 4F 96 03 01 72 45 5C 9A E0 60 CC E4 4A
The message MAC_A=HMAC (K_A, 0x01 || ID_A || ind || salt || u_1 || u_2)
from the subject A takes the following value:
The message $\text{MAC}_B=\text{HMAC}(\text{K}_B, 0x02 \parallel \text{ID}_B \parallel \text{ind} \parallel \text{salt} \parallel u_1 \parallel u_2)$ from the subject B takes the following value:

$\text{MAC}_B = \text{79 D5 54 83 FD 99 B1 2B CC A5 ED C6 BB E1 D7 B9 15 CE 04 51 B0 89 1E 77 5D 4A 61 CB 16 E3 3F CC}$

A.2.3 Curve id-GostR3410-2001-CryptoPro-C-ParamSet

The input protocol parameters in this example take the following values:

$N = 1$

$\text{ind} = 1$

$\text{ID}_A:$

$00 00 00 00$

$\text{ID}_B:$

$00 00 00 00$

$\text{PW}:$

$31 32 33 34 35 36 (\text{"123456")}$

$\text{salt}:$

$29 23 \ BE \ 84 \ E1 \ 6C \ D6 \ AE \ 52 \ 90 \ 49 \ F1 \ F1 \ BB \ E9 \ EB$

$Q_{\text{ind}}:$

$X = 0x1E36383E43BB6CFA2917167D71B7B5DD3D6D462B43D7C64282AE67DFBEC2559D$

$Y = 0x137478A9F721C73932EA06B45CF72E37EB78A63F29A542E563C614650C8B6399$

The function $F(\text{PW}, \text{salt}, 2000)$ takes the following values:

$F(\text{PW},\text{salt},2000):$

$BD \ 04 \ 67 \ 3F \ 71 \ 49 \ B1 \ 8E \ 98 \ 15 \ 5B \ D1 \ E2 \ 72 \ 4E \ 71$

$D0 \ 09 \ 9A \ A2 \ 51 \ 74 \ F7 \ 92 \ D3 \ 32 \ 6C \ 6F \ 18 \ 12 \ 70 \ 67$

The coordinates of the point $Q_{\text{PW}}$ are:

$X = 0x945821DAF91E158B839939630655A3B21FF3E146D27041E86C05650EB3B46B59$

$Y = 0x3A0C2816AC97421FA0E79605F17F0C9C3EB734CFF196937F6284438D70BDC48$

During the calculation of the message $u_1$ on the subject A the parameter $\alpha$, the point $\alpha*P$ and the message $u_1$ take the following values:

$\alpha = 0x3A54AC3F19AD9D0B1EAC8ACDCEA70E581F1DAC33D13FEAFD81E762378639C1A8$

$\alpha^*P:$

$X = 0x96B7F09C49D297C257A7DA48364C0076E59E48D221CBA604AE111CA3933B446A$

$Y = 0x54E4953D86B77ECCEB578500931E822300F7E091F79592CA202A020D762C34A6$

$u_1:$

$X = 0x81BBD6FCA464D2E2404A66D786CE4A777E739A89AE86C2DAC99D53273B75387$

$Y = 0x6B6DBD922EA7E06098F8B30A6EF07AD2EC86B2BF66391D82A30612EADD411$

During processing a message $u_1$, calculation the $K_B$ key and the message $u_2$ on the subject B the parameters $\beta$, $\text{src}$, $K_B = \text{HASH}($src$)$, $\beta^*P$ and $u_2$ take the following values:

$\beta = 0x448781782BF7C0E52A1DD9E6758FD3482D90D3CFCCCF42232CF357E59A4D49FD4$

$\text{src}:$

$16 \ A1 \ 2D \ 88 \ 54 \ 7E \ 1C \ 90 \ 06 \ BA \ A0 \ 08 \ E8 \ CB \ EC \ C9$

$D1 \ 68 \ 91 \ ED \ C8 \ 36 \ CF \ B7 \ 5F \ 8E \ B9 \ 56 \ FA \ 76 \ 11 \ 94$

$D2 \ BE \ 25 \ DA \ D3 \ 81 \ 8D \ 16 \ 3C \ 49 \ 4B \ 05 \ 9A \ 8C \ 70 \ A5$

$A1 \ B8 \ 8A \ 7F \ 80 \ A2 \ EE \ 35 \ 49 \ 30 \ 18 \ 46 \ 54 \ 2C \ 47 \ 0B$

$K_B:$

$BE \ 7E \ 7E \ 47 \ B4 \ 11 \ 16 \ F2 \ C7 \ 7E \ 3B \ 8F \ CE \ 40 \ 30 \ 72$

$CA \ 82 \ 45 \ 0D \ 65 \ DE \ FC \ 71 \ A9 \ 56 \ 49 \ E4 \ DE \ EA \ EC \ EE$

$\beta^*P:$
During processing a message $u_2$ and calculation the key on the subject A the $K_A$ key takes the following value:

$$K_A: \begin{array}{c} \text{BE 7E 7E 47 B4 11 16 F2 C7 7E 3B 8F CE 47 B4 11 16 F2 C7 7E} \\ \text{CA 82 45 0D 65 DE FC 71 A9 56 49 E4 DE EA EC EE} \end{array}$$

The message $MAC_A = HMAC (K_A, 0x01 || ID_A || ind || salt || u_1 || u_2)$ from the subject A takes the following value:

$$MAC_A: \begin{array}{c} \text{D3 B4 1A E2 C9 43 11 36 06 3E 6D 08 A6 1B E9 63} \\ \text{BD 5E D6 A1 FF F9 37 FA 8B 09 0A 98 E1 62 BF ED} \end{array}$$

The message $MAC_B = HMAC (K_B, 0x02 || ID_B || ind || salt || u_1 || u_2)$ from the subject B takes the following value:

$$MAC_B: \begin{array}{c} \text{D6 B3 9A 44 99 BE D3 E0 4F AC F9 55 50 2D 16 B2} \\ \text{CB 67 4A 20 5F AC 3C D8 3D 54 EC 2F D5 FC E2 58} \end{array}$$

A.2.4 Curve id=tc26-gost-3410-2012-512-paramSetA

The input protocol parameters in this example take the following values:

$$N = 1$$

$$\text{ind = 1}$$

$$\text{ID_A: 00 00 00 00}$$

$$\text{ID_B: 00 00 00 00}$$

$$\text{PW: 31 32 33 34 35 36 ('123456')}$$

$$\text{salt: 29 23 BE 84 E1 6C D6 AE 52 90 49 F1 F1 BB E9 EB}$$

$$\text{Q-ind: X = 0x2A17F8833A279532748871B5C5E88AEFB91126C64B8327289BEA62559425}$$
$$\text{D18198F133F400874328B220C74497CD240586CB249E158532CB8090776CD1C}$$
$$\text{Y = 0x728F0C4A73B48DA41CE928358FAD26B47A6E094E9362B7E82559F83C0DC4EC3A}$$
$$\text{4676BD3707DEEAF4CD85E99695C64C241ED6C22BE87DC0CF875F14367F723C5}$$

The function $F (PW, salt, 2000)$ takes the following values:

$$F(PW,salt,2000): \begin{array}{c} \text{BD 04 67 3F 71 49 B1 8E 98 15 5B D1 E2 72 4E 71} \\ \text{D0 09 9A A2 51 74 F7 92 D3 32 6C 6F 18 12 70 67} \\ \text{1C 62 13 E3 93 0E FD DA 26 45 17 92 C6 20 81 22} \\ \text{EE 60 D2 00 52 6D 69 5D FD 9F 5F OF D5 AB A7 02} \end{array}$$

The coordinates of the point $Q_{PW}$ are:

$$\text{X = 0x0C0AB53D0E09C607CAD758F558915A07DC5C87B45E9A58FDDF30EC3385960}$$
$$\text{283E030CD3229E46B07637785FD49D2CD711F46807A24C0AF9A42CEB2D740}$$
$$\text{Y = 0xDF93A8012B63A3D4F8A4D487DA15FC739EB31B20B3B0E8C80C32AAFB072CG3}$$
$$\text{37CF75B404719E5B4407C41D9A3216A08CA69C271484E9ED72BBAA52E28BBB}$$

During the calculation of the message $u_1$ on the subject A the parameter alpha, the point alpha*P and the message $u_1$ take the following values:

$$\text{alpha=0x3CE54235DB52FE798824AEAD11BB16FA766857D04A4AF7D468672F16D90E7396}$$
$$\text{046A46F815693E85B1CE5464DA9270181F82333B0715057BBE8D61D400505F0E}$$
alpha*P:

\[
X = 0xB93093EB0FCC463239B7DF276E09E592FCFC9B635504EA4531655D76A0A3078E \\
2B4E51CFE2FA400CC5DE9F8E369DB204B3E8ED7EDD85E5CCA654C1AED70E396 \\
Y = 0x809770B8D910EA30BD2FA89736E91DC31815D2D9B31128077EEDC371E9F69466 \\
F497DC6D5B51FAD5C87F860EE256109138C4A9CD96B262E65A8F590520FC882 \\
\]

u_1:

\[
X = 0xE7510A9EDD37B869566C81052E2515E1563FDFE79F1D782D6200F3C3CC2764D \\
4D0070B73AD5A47BAE9A8F2289C1B07DAC26A1A2FF9D3EBC0A8A494F179F13 \\
Y = 0xBA33B912570777B626A5337BC7F727952460E0B2775707F4537372E902DF5 \\
636080B25399751BF48BF1543C3C2319A1957C23F39F89EF54A8F043853F82DE \\
\]

During processing a message u_1, calculation the K_B key and the message u_2 on the subject B the parameters betta, src, K_B = HASH(src), betta*P and u_2 take the following values:

betta=0xB5C286A79AA8E97EC0E19BC1959A1D15F12F8C97870B9A6D8CC12811A56A3BB1 \\
144061082579649D468CD9C2D2D76598A27973D5960C5F50BCE28D8345F4 \\

src:

84 59 C2 0C B5 C5 32 41 6D B9 28 EB 50 C0 52 0F \\
B2 1B 9C D3 9A 4E 76 06 B2 21 BE 15 CA 1D 02 DA \\
08 15 DE C4 49 79 C0 8C 7D 23 07 AF 24 7D DA 1F \\
89 EC 81 20 69 F5 D9 CD E3 06 AF F0 BC 3F D2 6E \\
D2 01 B9 53 52 A2 56 06 B6 43 E8 88 30 2E FC 8D \\
3E 95 1E 3E B4 68 4A DB 5C 05 7B 8F 8C 89 B6 CC \\
0D EE D1 00 06 5B 51 8A 1C 71 7F 76 82 FF 61 2B \\
BC 79 8E C7 B2 49 0F B7 00 3F 94 33 87 37 1C 1D \\
K_B:

53 24 DE F8 48 B6 63 CC 26 42 2F 5E 45 EE C3 4C \\
51 D2 43 61 B1 65 60 CA 58 A3 D3 28 45 86 CB 7A \\
betta*P:

\[
X = 0x238B38644E440452A99FA6B93D9FD7DA0CB83C32D3C1E3CFE5DF5C3EB0F9DB91 \\
E588DAEDC949EA2FB867AE855A21B4077353C0794716A6480995113D8C20C7AF \\
Y = 0xB2273D5734C1897FD815A7008862938C87C47A7E877423D95243EB7EDB02FD2 \\
C456CF9FC956F078A59AA86F19DD1075E5167E4ED35208718EA93161C530ED14 \\
\]

u_2:

\[
X = 0xC384A126216E81B372001E77C1FE9C7547F9223CF7BB865C4472EC18BE079A \\
678C5C5AE4028EF3620CCE535514F1E589F8A0C43SC3AECFBD2EE8784D953411 \\
Y = 0x8B520D083A4AF257E8A54EC90CBADBAF4FEED2C2D868C82FF04FCBB9EF638E56 \\
F6BAF9472D477414DA7E36F538ED223D2E02FAE1A20A98C59FCF03B6F30D \\
\]

During processing a message u_2 and calculation the key on the subject A the K_A key takes the following value:

K_A:

53 24 DE F8 48 B6 63 CC 26 42 2F 5E 45 EE C3 4C \\
51 D2 43 61 B1 65 60 CA 58 A3 D3 28 45 86 CB 7A \\

The message MAC_A=HMAC (K_A, 0x01 || ID_A || ind || salt || u_1 || u_2) from the subject A takes the following value:

MAC_A:

E8 EF 9E A8 F1 E6 B1 26 68 E5 8C D2 2D D8 EE C6 \\
4A 16 71 00 39 FA A6 B6 03 99 22 20 FA FE 56 14 \\
The message MAC_B=HMAC (K_B, 0x02 || ID_B || ind || salt || u_1 || u_2) from the subject B takes the following value:

MAC_B:

61 14 34 60 83 6B 23 5C EC D0 B4 9B 58 7E A4 5D \\
51 3C 3A 38 78 3F 1C 9D 3B 05 97 0A 95 6A 55 BA
A.2.5 Curve id-tc26-gost-3410-2012-512-paramSetB

The input protocol parameters in this example take the following values:

\[ N = 1 \]
\[ \text{ind} = 1 \]
\[ \text{ID}_A: \quad 00 \ 00 \ 00 \ 00 \]
\[ \text{ID}_B: \quad 00 \ 00 \ 00 \ 00 \]
\[ \text{PW}: \quad 31 \ 32 \ 33 \ 34 \ 35 \ 36 \ ('123456') \]
\[ \text{salt}: \quad 29 \ 23 \ BE \ 84 \ 1E \ 56 \ AE \ 52 \ 90 \ 49 \ F1 \ BF \ EB \]
\[ Q_{\text{ind}}: \]
\[ X = 0x7E1FAE825E035BEC244BEF2D0E5E5BF436633CF50E55231DEA9C9CF21D4C8C33 \]
\[ DF5D4305DE92971F04A4BC07E00D87DBD720E6B66E49079285AFAF120E171149 \]
\[ Y = 0x2C89998B787D463805BA0D858A196592DB20AB161558FF724F7A8572D3209 \]
\[ 53967AE821AFDEAE99B77C83A2528EF6F6E02F68BDA4679D7F2704947DBC408 \]

The function \( F(\text{PW}, \text{salt}, 2000) \) takes the following values:

\[ F(\text{PW}, \text{salt}, 2000): \]
\[ BD \ 04 \ 67 \ 3F \ 71 \ 49 \ B1 \ 8E \ 98 \ 15 \ 5B \ D1 \ E2 \ 72 \ 4E \ 71 \]
\[ D0 \ 09 \ 9A \ 2A \ 51 \ 74 \ F7 \ 92 \ D3 \ 32 \ 6C \ 6F \ 18 \ 12 \ 70 \ 67 \]
\[ 1C \ 62 \ 13 \ E3 \ 93 \ 0E \ FD \ DA \ 26 \ 45 \ 17 \ 92 \ C6 \ 20 \ 81 \ 22 \]
\[ EE \ 60 \ D2 \ 00 \ 52 \ OD \ 69 \ 5D \ FD \ 9F \ 5F \ 0F \ D5 \ AB \ A7 \ 02 \]

The coordinates of the point \( Q_{\text{PW}} \) are:

\[ X = 0x7D03E65B8050D1E12CBB601A17B9273B0E728F5021CD47C8A44D822E4627BA5F \]
\[ 9C696286A2CDDA9A065590366B4DED6DC4A1849604AD54F87A60AFA621161 \]
\[ Y = 0x16037DAD45421EC50B00D50BDC6AC3B8534BC1D3A2F85DB27C3373580FEF87C \]
\[ 2C743B7ED30F2BE22958044E716F93A61CA3213A361A2797A16A3AE62957377 \]

During the calculation of the message \( u_1 \) on the subject A the parameter \( \alpha \), the point \( \alpha*P \) and the message \( u_1 \) take the following values:

\[ \alpha = 0x715E893FA639BF341296E0623E629DADFH26B163C278767A7982A989462A3863 \]
\[ FE12AEF843D03D59C4DC4720570D4163DB0805C7C10C4E418F9CB785B04B997 \]
\[ \alpha^*P: \]
\[ X = 0x10C479EA1C04D32C02B05769AC42D96226FF033C1191436777F66916030D87D \]
\[ 02F993738ED7699D7619FFCE7C1F3C4DB5E5DF49E2186D6FAE12EB5767602B9 \]
\[ Y = 0x039F604419140E707F26D59D979136A831CCE43E1C5F6000D1DDF839D0CA3D \]
\[ 52FBD943BF04DDCED1AA2EC85E89EB7487ACDEF239C07D15084D796784F35436 \]
\[ u_1: \]
\[ X = 0x45C05CCEE8290762F2470B179B4306D2B2911CEB1447F72EF11D0498CE7E21 \]
\[ FF163FE72044B4E732AD8CBE3C31217820F53A60726315BCEB5BC6DA5CF1E0 \]
\[ Y = 0x5BE483E382D05F7048CF6A5045D99E62755B5ACC9554EC4A5B2093E12A2DD \]
\[ 5C6066BC9EDE39373BA189920808B419E38B39BBDEDE0B09A5CAEA984D02E \]

During processing a message \( u_1 \), calculation the K_B key and the message \( u_2 \) on the subject B the parameters \( \beta \), \( \text{src} \), \( K_B = \text{HASH}(\text{src}) \), \( \beta^*P \) and \( u_2 \) take the following values:

\[ \text{betta} = 0x30FA8C2B4146C2DBBE82BED04D7378877E8C06753BD0A0FF71EBF2BEFE8DA8F3 \]
\[ DC083646E82ECE7C5C961281B6505140F8407413F03C2CB1D201EA1286CE30E6D \]
\[ \text{src}: \]
\[ 3F \ 04 \ 02 \ E4 \ 0A \ 9D \ 59 \ 63 \ 20 \ 5B \ CD \ F4 \ FD \ 89 \ 77 \ 91 \]
\[ 9B \ BA \ F4 \ 80 \ F8 \ E4 \ FB \ D1 \ 25 \ 5A \ EC \ E6 \ ED \ 57 \ 26 \ 4B \]
\[ D0 \ A2 \ 87 \ 98 \ 4F \ 59 \ D1 \ 02 \ 04 \ B5 \ F4 \ 5E \ 4D \ 77 \ F3 \ CF \]
\[ 8A \ 63 \ B3 \ 1B \ EB \ 2D \ F5 \ 9F \ 8A \ F7 \ 3C \ 20 \ 9C \ CA \ 8B \ 50 \]
\[ B4 \ 18 \ D8 \ 01 \ E4 \ 90 \ AE \ 13 \ 3F \ 04 \ F4 \ F3 \ F4 \ D8 \ FE \ 8E \]
During processing a message \( u_2 \) and calculation the key on the subject \( A \) the \( K_A \) key takes the following value:

\[
K_A: \\
D5 90 E0 5E F5 AE CE 8B 7C FB FC 71 BE 45 5F 29 \\
A5 CC 66 6F 85 CD B1 7E 7C C7 16 C5 9F F1 70 E9
\]

The message \( MAC_A = \text{HMAC}(K_A, 0x01 || ID_A || \text{ind} || salt || u_1 || u_2) \) from the subject \( A \) takes the following value:

\[
MAC_A: \\
DE 46 BB 4C 8C E0 8A 6E F3 B8 DF AC CC 1A 39 B0 \\
8D 8C 27 B6 CB 0F CF 59 23 86 A6 48 F4 E5 BD 8C
\]

The message \( MAC_B = \text{HMAC}(K_B, 0x02 || ID_B || \text{ind} || salt || u_1 || u_2) \) from the subject \( B \) takes the following value:

\[
MAC_B: \\
EC B1 1D E2 06 1C 55 F1 D1 14 59 CB 51 CE 31 40 \\
99 99 2F CA 22 2F B1 4F CE AB 96 EE 7A AC
\]

A.2.6 Curve id-tc26-gost-3410-2012-256-paramSetA

The input protocol parameters in this example take the following values:

\[
N = 1 \\
\text{ind} = 1 \\
ID_A: \\
00 00 00 00 \\
ID_B: \\
00 00 00 00 \\
PW: \\
31 32 33 34 35 36 (\text{123456}) \\
salt: \\
29 23 BE 84 E1 6C D6 AE 52 90 49 F1 F1 BB E9 EB \\
Q, ind: \\
X = 0xB51ADF93A40AB15792164FAD3352F95B66369EB2A4EF5EFAE32829320363350E \\
Y = 0x74A358CC08593612F5955D249C96AFB7E8B0BB6D8BD2BEE491046650D822BE18 \\
The function \( F(PW, \text{salt}, 2000) \) takes the following values:

\[
F(PW, \text{salt}, 2000): \\
BD 04 67 3F 71 49 B1 8E 98 15 5B D1 E2 72 4E 71 \\
D0 09 9A A2 51 74 F7 92 D3 32 6C 6F 18 12 70 67 \\
The coordinates of the point \( Q, PW \) are:

\[
X = 0xDBF9927078956812FA48C6E695DF589DEF1D18A2D4D35A96D75BF6854237629
\]
During the calculation of the message $u_1$ on the subject A the parameter alpha, the point $\alpha P$ and the message $u_1$ take the following values:
\[
\alpha = 0x147B72F6684FB6D1B418A899F7DBECAF5FCE60B13685BAA9532654A7F0707F
\]
\[
\alpha P:
X = 0x33FBAC14EAE538275A769417829C431BD9FA622B6F02427EF55BD60EE6BC888
Y = 0x22F2EBCF960A82E6CDB402D3DADA511B2FBA925383C2273D952EA2D406EA46
\]
\[
u_1:
X = 0xE569AB544E3A13C41077DE97D659A1B7A13F61DDD808B633A5621FE258A2C43
Y = 0xA21A43A08F4D71669297ECD6F86553A808925BF34802BF7EC34C548A40B2C0
\]

During processing a message $u_1$, calculation the $K_B$ key and the message $u_2$ on the subject B the parameters betta, src, $K_B = \text{HASH(src)}$, betta$P$ and $u_2$ take the following values:
\[
betta = 0x30D5CFADAA0E31B405E6734C03EC4C5DF0F02F4B2A259A3B320EE6453567B4CB
\]
\[
\text{src:}
A3 39 A0 9C EF 1A 6F FD 4C A1 28 04 9E 06 84
DF 4A 97 75 B6 89 A3 37 84 1B F7 D7 91 20 7F 35
11 86 28 F7 28 8E AA 0F 7E C8 1D A2 0A 24 FF 1E
69 93 C6 3D 9D D2 6A 90 B7 4D D1 A2 66 28 06 63
\]
\[
K_B:
7D F7 1A C3 27 ED 51 7D BD 4E 03 E8 17 C6 20 4B
C1 91 65 B9 D1 00 2B 9F 10 88 A6 CD A6 EA CF 27
\]
\[
betta P:
X = 0x252D89FAB735433970564F2F28CFA1B57D640CB902BC633A53BF44155022CB2
Y = 0x10EF6A82E9F170F9423AA81D6B4CE5DE0D4947512962874870E6F2849A96F
\]
\[
u_2:
X = 0x190D2F283F7E61065DB3227D7FBDF429CEBF93791262CB29569BD63C86CA4
Y = 0xB3F1715721E921897CCDE046C9B43A8386DBF7818A112F15A02BC820AC8F6D
\]

During processing a message $u_2$ and calculation the key on the subject A the $K_A$ key takes the following value:
\[
K_A:
7D F7 1A C3 27 ED 51 7D BD 4E 03 E8 17 C6 20 4B
C1 91 65 B9 D1 00 2B 9F 10 88 A6 CD A6 EA CF 27
\]

The message $\text{MAC}_A = \text{HMAC} (K_A, 0x01 || \text{ID}_A || \text{ind} || \text{salt} || u_1 || u_2)$ from the subject A takes the following value:
\[
\text{MAC}_A:
F9 29 B6 1A 3C 83 39 85 B8 29 F2 68 55 7F A8 11
00 9F 82 0A B1 A7 30 B5 AA 33 4C 3E 6B A3 17 7F
\]

The message $\text{MAC}_B = \text{HMAC} (K_B, 0x02 || \text{ID}_B || \text{ind} || \text{salt} || u_1 || u_2)$ from the subject B takes the following value:
\[
\text{MAC}_B:
A2 92 8A 5C F6 20 BB C4 90 0D E4 03 F7 FC 59 A5
E9 80 B6 8B E0 46 D0 B5 D9 B4 AE 6A BF A8 0B D6
\]

A.2.7 Curve id-tc26-gost-3410-2012-512-paramSetC

The input protocol parameters in this example take the following values:
\[
N = 1
\]
\[
\text{ind} = 1
\]
\[
\text{ID}_A:
00 00 00 00
\]
\[
\text{ID}_B:
00 00 00 00
\]

PW:
salt:
29 23 BE 84 E1 6C D6 AE 52 90 49 F1 BB E9 EB

Q_ind:

X = 0x489C91784E02E98F19A803ABC319917F37689E5A18965251CE2FF4E8D8B298F5B5A7470F9E0E713487F96FA48397B3D09270C9D367EB5EOE661ADEEB51581D
Y = 0x684EA885ACA6A4EAF1B3FE36C0852A3BE3BD8011B0EF18E203FF87028D6EB5DB2C144A0DCC7126542BFDF72CA2A43FA4F4939DA66D9A60793C704A8C94E16F18

The function F (PW, salt, 2000) takes the following values:

F(PW,salt,2000):

BD 04 67 3F 71 79 B1 8E 98 15 5B D1 E2 72 4E 6F 18 12 70 67
1C 62 13 E3 93 0E FD DA 26 45 17 92 C6 20 81 22
EE 60 D2 00 52 0D 69 5D FD 9F 5F 0F D5 AB A7 02

The coordinates of the point Q_PW are:

X = 0x0185AE6271A81BB7F236A955F7CAA26FB63849813C0287D96C83A15AE6B6A86467AB136BDB88CE8C7DC2E5B97F5F2BFA2C21082FA3CF3DB5513C3E6D7D210E8Y = 0xED0220F92EF771A71C64CECC77986DB7C03D37B3E2ABE383F3CE5E074A762EC082539E2102B87532661275C4B1D16D2789CDAABC58ACD7F381DE70AB64F9B8

During the calculation of the message u_1 on the subject A the parameter alpha, the point alpha*P and the message u_1 take the following values:

alpha=0x332F930421D14CFE260042159F18E49FD5A54167E94108AD80B1DE60B13DE7999A34D611E63FF3F870E5110247DF8EC7466E64ACF385E52CB89AF19EFDF0alpha*P:

X = 0x561655966D52952E805574F4281F1ED3A2D498932B00CBA9DECB42837F09835BFFBFE2D4BD6B242F7E85F792E1A6F2413E12DDD63E4437E13D72693469ADY = 0xF6B18328B2715BDE7F4178615273A36135BC0BF62F7DBB9F80164AD3647AD03660F1806C64691BADEF30F793720F8E3FEAE639D16A5A4C372DCBF80E82u_1:

X = 0x40645B4B9A90874DE89866A336F98BAE6ADA41C9B7594A335E4A16486C553C7C35CD4797AB5B4340BFC70C4AC101B16A90A14A175E5B9B543D5151CB7Y = 0x267FBB18D0B79559D1875909F2A15F7B49ECD8ED166CF7F4CDF1F448915504835E80D52E8B34ADA05B5E87C92791B1CFE8F5048DC443AA0983AA19192B8407During processing a message u_1, calculation the K_B key and the message u_2 on the subject B the parameters betta, src, K_B = HASH(src), betta*P and u_2 take the following values:
betta=0x384B1777E7D054F96212686B613881880BD8A6C892DCC656178F014D2C093432A033EE10415F13A16D44C2AD61E62E05A7F7EC286BCEA3EA4D453F8634FA2src:

4F 4D 64 B5 D0 70 08 E9 E6 85 87 4F 88 2C 3E 1E 60 A6 67 5E ED 42 1F C2 34 16 3F DE B4 4C 69 18 B7 BC CE AB 88 A0 F3 FB 78 8D A8 DB 10 18 51 FF 1A 41 68 22 BA 37 C3 53 CE C4 C5 A5 23 95 B7 72 AC 93 C0 54 E3 F4 05 5C ED 6F F0 BE E4 A6 A2 4E D6 8B 86 FE FA 70 DE 4A 2B 16 08 51 42 A4 DF F0 5D 32 EC 7D DF E3 04 F5 C7 04 FD FA 06 0F 64 E9 E8 32 14 00 25 F3 92 E5 03 50 77 0E 3F B6 2C AC K_B:

A0 B3 84 A6 2F 4B E1 AE 48 98 FC FA 3A 6D AA 3F AA 45 1B 3E C5 B9 E3 75 F8 9E 92 9F 4B 13 25 8Cbetta*P:

X = 0xB7C5818687083433BC1AFF61CB5CA79E3823025E0C1F123B8651E62173CE6873F3E6FF7281C2E45F4524F66B0C263616ED08FD210AC4355CA3292B51D71C3
Y = 0x497F1405DB89BDAF502ED3B1429AD307773102B5E568070F016A4E0
    C76DB0B68AC23FBD6DE6754A4DF591EB18BA0D48DF7AA40973A2F1FC8A55
u_2:
X = 0xB772FD97D6FDEC1DA0771BC059B3E5ADF9858311031AE5AE6B0C8104B4105
    C45AA6C0689A8E636C87DB62CC0AFC9A48CA6E381286CC73F37E1DD04F445
Y = 0xC64F69425FFEB2995130E85A08EDC3A68EC28EE68469F7F9098D0BCBDD843AC
    573578DA6BA1CB3F5F069F205233853F60255C4B28586C9A164357497B1018C

During processing a message u_2 and calculation the key on the subject A
the K_A key takes the following value:
K_A:
  A0 83 84 A6 2F 4B E1 A8 48 98 FC A3 6D AA 3F AA
  45 1B 3E C5 9C E3 75 95 92 91 4F 13 25 8C
The message MAC_A=HMAC (K_A, 0x01 || ID_A || ind || salt || u_1 || u_2)
from the subject A takes the following value:
MAC_A:
  12 63 F2 89 0E 90 EE 42 6B 9A 0B 8A B9 EA 7F 1F
  FF 26 E1 60 5C C6 5D E2 FF 96 91 E3 95 82 76 87
The message MAC_B=HMAC (K_B, 0x02 || ID_B || ind || salt || u_1 || u_2)
from the subject B takes the following value:
MAC_B:
  6D FD 06 04 5D 6D 9A 04 E4 19 B0 0E 00 35 B9 D2
  E3 AB 09 8B 7C A4 AD 52 54 60 FA B6 21 85 AA 57

Appendix B. Point Verification Script

The points from the Appendix A.1 were generated with the following point verification script in Python:

```python
curvesParams = [
    
    "OID":"id-GostR3410-2001-CryptoPro-A-ParamSet",
    "p":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD97,
    "a":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD94,
    "b":166,
    "m":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD94,
    "x":1,
    "y":0xB772FD97D6FDEC1DA0771BC059B3E5ADF9858311031AE5AE6B0C8104B4105,
    "n":32
],

"OID":"id-GostR3410-2001-CryptoPro-B-ParamSet",
"p":0x8000000000000000000000000000000000000000000000000000000000000C99,
"a":0x8000000000000000000000000000000000000000000000000000000000000C99,
"b":0x3E1A419A269A5F866A7D3C25C3DF80AE97259373FF2B182F49D4CE7E1BCB8B,
"m":0x8000000000000000000000000000000000000000000000000000000000000C96,
"q":0x8000000000000000000000000000000000000000000000000000000000000C96,
"x":1,
"y":0x3FA8124359F6680B83D1C3EB2C070E5C545C9858D03ECF744BF8D717717EFC,
"n":32
],

"OID":"id-GostR3410-2001-CryptoPro-C-ParamSet",
```
"p":0x9B9F605F5A858107AB1EC85E6B41C8AACF846E86789051D37998F7B9022D759B,
"a":0x9B9F605F5A858107AB1EC85E6B41C8AACF846E86789051D37998F7B9022D759B,
"b":32858,
"m":0x9B9F605F5A858107AB1EC85E6B41C8AACF846E86789051D37998F7B9022D759B,
"o":0,
"y":0x41ECE55743711A8C3CBF3783CD08C0EE4D4DC440D4641A8F366E550DFDB3BB67,
"n":32
},

{"OID":"id-tc26-gost-3410-2012-512-paramSetA",
"p":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296|0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<80|0xFFFFFFFFFFFFFFFFFDC7L,
"a":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296|0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<80|0xFFFFFFFFFFFFFFFFFDC4L,
"b":0x8C2505DEDFC86DDC1BD0B2B6667F1DA34B82574761CB0E879BD08L<<296|0x1CFD0B6265EE3CB090F30D27614CB4574010DA90DD62EF9D4EBEEL<<80|0x4761503190785A71C760L,
"m":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296|0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<80|0xB85DCACDB1411F10B275L,
"q":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296|0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<80|0xB85DCACDB1411F10B275L,
"x":3,
"y":0x7503C7E87A36AE3A61B8816E25450E6CE5E1C93ACF1ABC1778064L<<296|0xFDCEBEF921DF1626BE4FD036E93D735E6A50E3A41E98028FE5FC235L<<80|0xF5B889A589CB5215F2A4L,
"n":64
},

{"OID":"id-tc26-gost-3410-2012-512-paramSetB",
"p":0x8000000000000000000000000000000000000000000000000000L<<296|0x8000000000000000000000000000000000000000000000000000L<<80|0x00000000006FL,
"a":0x8000000000000000000000000000000000000000000000000000L<<296|0x8000000000000000000000000000000000000000000000000000L<<80|0x0000000000000000000000000000000000000000000000000000L,
"b":0x687D1B459DC84145E3E06CF65E2517B97C7D614AF138BDBF85DCL<<296|0x806C4B289F3E9652DB1416D217F8B276FAD1AB69C50F78BEE1FA3L<<80|0x106EFB8CCBC7C5150116L,
"m":0x8000000000000000000000000000000000000000000000000000L<<296|0x8000000000149A1EC142565A54ACFDB77BD9D40CFA8B996712101BL<<80|0x0EA0EC6346C54374F25BDL,
"q":0x8000000000000000000000000000000000000000000000000000L<<296|0x8000000000149A1EC142565A54ACFDB77BD9D40CFA8B996712101BL<<80|0x0EA0EC6346C54374F25BDL,
"x":2,
"y":0x1A8F7EDA389B094C2C071E3647A8940F3C123B697578C213BE6DD9L<<296|0xE6C8EC735DCB228FD1EDF4A39152CBA58C0398828041055F94CL<<80|0xEEEC7E21340780FE41BDL,
def str2list( s ):
    res = []
    for c in s:
        res += [ ord( c ) ]
    return res

def list2str( l ) :
    r = ""
    for k in l:
        r += chr( k )
    return r

def hprint( data ):
r = ""
for i in range( len( data ) ):
    r += "%02X " % data[ i ]
    if i % 16 == 15:
        r += "n"
print( r )

class Stribog:

    __A = [
    0x8e20faa72ba0b470, 0x47107ddd9b505a38, 0xad08b0e0c3282d1c,
    0xd8045870ef14980e, 0x6c022c38f90a4c07, 0x1b8e0b798c13c8,
    0x83478b07b2468764, 0xa011d380818e8f40, 0x5086e740ce47c920,
    0x2843fd2067adea10, 0x14aff010bdd87508, 0x0ad97808d06cb404,
    0x05e23c0468365a02, 0xc711203142b2d01, 0x46b0f011a83988e,
    0x90dab52a387ae76f, 0x486dd415c3dfdb9, 0x24b86a940e90f0d2,
    0x125c354207487869, 0x092e94218d43cb, 0x8a1749ec8121e5d,
    0x4585254f64090fa0, 0xacc9ca9328a8950, 0x9d4df05d5f661451,
    0xc0a878a0a1330aa6, 0x60543c50de970553, 0x302a1268fc58ca7,
    0x1815014b9ec46dd, 0x0c84890ad27632e0, 0x0642ca05693b9f70,
    0x321658c9a3c138, 0x86275df09ce8aa8, 0x439da784e745554,
    0x1af503c273aa42a, 0x9d60281e91d5215, 0xe230140cf092984,
    0x71180a38609a42, 0xb60c05ca3024d1, 0x5b068ce51810a9e,
    0x456c34887a3805b9, 0xac361a443d18c6d, 0x561b0d22900e4669,
    0x2b38811480723ba, 0x9b81f486248d95d, 0xc3e9224312c81a0,
    0x1f5f0e964ee50, 0xf97d86d98a327728, 0x4f1a025a880b329c,
    0x727d012a548b194e, 0x39b008152acb827, 0x9258048415b419d,
    0x492c02428fbaec0, 0xaa16012142f35760, 0x550b8e9e21f7a530,
    0xa48b4749ef5dc11, 0x70a6a56e2440598e, 0x3853dc371220a247,
    0x71a76e9501051ad, 0x0edd37c48a08a6d8, 0x70e9065450436c,
    0x8d70c431ac02a736, 0xc838629656501d1b, 0x641c3142b8ee083
    ]

    __Sbox = [
    0xFC, 0xEE, 0xDD, 0x11, 0xCF, 0x6E, 0x31, 0x16, 0xFB, 0xC4, 0xFA,
    0xDA, 0x23, 0xC5, 0x04, 0xD4, 0xE9, 0x77, 0xF0, 0xDB, 0x93, 0x2E,
    0x99, 0xBA, 0x17, 0x36, 0x0F1, 0xBB, 0x14, 0xCD, 0x5F, 0xC1, 0xF9,
    0x18, 0x65, 0x5A, 0xE2, 0x5C, 0xEF, 0x21, 0x81, 0x1C, 0x3C, 0x42,
    0x8B, 0x01, 0x8E, 0x4F, 0x05, 0x84, 0x02, 0xAE, 0xE3, 0x6A, 0x8F,
    0xA0, 0x06, 0x0B, 0xED, 0x98, 0x7F, 0xD4, 0xD3, 0x1F, 0xEB, 0x34,
    0x2C, 0x51, 0xEA, 0xC8, 0x48, 0xAB, 0xF2, 0x2A, 0x68, 0xA2, 0xFD,
    0x3A, 0xCE, 0xCC, 0xB5, 0x70, 0x0E, 0x56, 0x08, 0x0C, 0x76, 0x12,
    0xBF, 0x72, 0x13, 0x47, 0x9C, 0xB7, 0x5D, 0x87, 0x15, 0xA1, 0x96,
    0x29, 0x10, 0x7B, 0x9A, 0xC7, 0xF3, 0x91, 0x78, 0x6F, 0x9D, 0x9E,
    0xB2, 0xB1, 0x32, 0x75, 0x19, 0x3D, 0xFF, 0x35, 0x8A, 0x7E, 0x6D,
    0x54, 0xC6, 0x80, 0xC3, 0xBD, 0x0D, 0x57, 0xDF, 0xF5, 0x24, 0xA9,
    0x3E, 0xA8, 0x43, 0xC9, 0xD7, 0x79, 0xD6, 0xF6, 0x7C, 0x22, 0xB9,
    0x03, 0xE0, 0x0F, 0xEC, 0xDE, 0x7A, 0x94, 0xB0, 0xBC, 0xDC, 0xE8,
    0x28, 0x50, 0x4E, 0x33, 0x0A, 0xA4, 0xA7, 0x97, 0x60, 0x73, 0x1E,
    0x00, 0x62, 0x44, 0x1A, 0xB8, 0x38, 0x82, 0x64, 0x9F, 0x26, 0x41,
    0xAD, 0x45, 0x46, 0x92, 0x27, 0x5E, 0x55, 0x2F, 0x8C, 0xA3, 0xA5,
    0x7D, 0x69, 0xD5, 0x95, 0x3B, 0x07, 0x58, 0xB3, 0x40, 0x86, 0xAC,
\( \tau = [\)
\[
0, 8, 16, 24, 32, 40, 48, 56,
1, 9, 17, 25, 33, 41, 49, 57,
2, 10, 18, 26, 34, 42, 50, 58,
3, 11, 19, 27, 35, 43, 51, 59,
4, 12, 20, 28, 36, 44, 52, 60,
5, 13, 21, 29, 37, 45, 53, 61,
6, 14, 22, 30, 38, 46, 54, 62,
7, 15, 23, 31, 39, 47, 55, 63
\]
\( \)
\( \gamma = [\)
\[
0xb1, 0x08, 0x5b, 0xda, 0x1e, 0xca, 0xda, 0xe9,
0xeb, 0xcb, 0x2f, 0x81, 0xc0, 0x65, 0x7c, 0x1f,
0x2f, 0x6a, 0x76, 0x43, 0x2e, 0x45, 0xd0, 0x16,
0x71, 0x4e, 0xb8, 0x8d, 0x75, 0x85, 0xc4, 0xfc,
0x4b, 0x7c, 0xe0, 0x91, 0x92, 0x67, 0x69, 0x01,
0xa2, 0x42, 0x2a, 0x08, 0xa4, 0x60, 0xd3, 0x15,
0xd0, 0x65, 0x59, 0xf2, 0xa6, 0x45, 0x07
\],
\[
0x6f, 0xa3, 0xb5, 0x8a, 0xa9, 0x9d, 0x2f, 0x1a,
0x4f, 0xe3, 0xd5, 0x0f, 0x70, 0xb5, 0xd7,
0xf3, 0xfe, 0xea, 0x72, 0xa0, 0x23, 0x2b, 0x98,
0x61, 0xd5, 0x5e, 0x0f, 0x16, 0xb5, 0x01, 0x31,
0x9a, 0xb5, 0x17, 0x6b, 0xd6, 0x99, 0x58,
0x5c, 0xb5, 0x61, 0xc2, 0xdb, 0xa0, 0xa7, 0xca,
0x55, 0xdd, 0xa2, 0x1b, 0xd7, 0xcb, 0xcd, 0x56,
0xe6, 0x79, 0x04, 0x70, 0x21, 0xb1, 0x9b, 0xb7
\],
\[
0xf5, 0x74, 0xdc, 0xac, 0xc7, 0xf2, 0xc7,
0x0a, 0x39, 0xfc, 0x28, 0x6a, 0x3d, 0x84, 0x35,
0x06, 0xf1, 0xe5, 0xe5, 0xe9, 0x0f, 0x8b,
0xf2, 0xea, 0x75, 0x14, 0xb1, 0x29, 0x7b, 0x7b,
0x3d, 0xe2, 0x0f, 0xe4, 0x90, 0x35, 0x9e, 0xb1,
0xc1, 0xc9, 0x3a, 0x37, 0x60, 0x62, 0xdb, 0x09,
0xc2, 0xb6, 0xf4, 0x43, 0x86, 0x7a, 0xdb, 0x31,
0x99, 0x1e, 0x96, 0xf5, 0xa0, 0xba, 0x0a, 0xb2
\],
\[
0xef, 0x1f, 0xdf, 0xb3, 0xe8, 0x15, 0x66, 0xdb,
0xf9, 0x48, 0xe1, 0xa0, 0x5d, 0x71, 0xe4, 0xdd,
def __AddModulo(self, A, B):
    result = [0] * 64
    t = 0
    for i in reversed(range(0, 64)):
        t = A[i] + B[i] + (t >> 8)
        result[i] = t & 0xFF
    return result

def __AddXor(self, A, B):
    result = [0] * 64
    for i in range(0, 64):
        result[i] = A[i] ^ B[i]
    return result

def __S(self, state):
    result = [0] * 64
    for i in range(0, 64):
result[i] = self.__Sbox[state[i]]
return result

def __P(self, state):
    result = [0] * 64
    for i in range(0, 64):
        result[i] = state[self.__ Tau[i]]
    return result

def __L(self, state):
    result = [0] * 64
    for i in range(0, 8):
        t = 0
        for k in range(0, 8):
            for j in range(0, 8):
                if ((state[i * 8 + k] & (1 << (7 - j))) != 0):
                    t ^= self.__A[k * 8 + j]
        for k in range(0, 8):
            result[i * 8 + k] = (t & (0xFF << (7 - k) * 8)) >> (7 - k) * 8
    return result

def __KeySchedule(self, K, i):
    K = self.__AddXor(K, self.__C[i])
    K = self.__S(K)
    K = self.__P(K)
    K = self.__L(K)
    return K

# E(K, m)
def __E(self, K, m):
    state = self.__AddXor(K, m)
    for i in range(0, 12):
        state = self.__S(state)
        state = self.__P(state)
        state = self.__L(state)
        K = self.__KeySchedule(K, i)
        state = self.__AddXor(state, K)
    return state

def __G_n(self, N, h, m):
    K = self.__AddXor(h, N)
    K = self.__S(K)
    K = self.__P(K)
    K = self.__L(K)
    t = self.__E(K, m)
    t = self.__AddXor(t, h)
    return self.__AddXor(t, m)

def __Padding(self, last, N, h, Sigma):
    if (len(last) < 64):
        padding = [0] * (64 - len(last))
        padding[-1] = 1
padded_message = padding + last
h = self.__G_n(N, h, padded_message)
N_len = [0] * 64
N_len[63] = (len(last) * 8) & 0xff
N_len[62] = (len(last) * 8) >> 8
N = self.__AddModulo(N, N_len)
Sigma = self.__AddModulo(Sigma, padded_message)
return (h, N, Sigma)

def digest(self, message, out=512):
    return list2str(self.GetHash(str2list(message), out))

def GetHash(self, message, out=512, no_pad=False):
    N = [0] * 64
    Sigma = [0] * 64
    if out == 512:
        h = [0] * 64
    elif out == 256:
        h = [0x01] * 64
    else:
        print("Wrong hash out length!")

    N_512 = [0] * 64
    N_512[62] = 0x02    # 512 = 0x200

    length_bits = len(message) * 8
    length = len(message)

    i = 0
    asd = message[::-1]
    while (length_bits >= 512):
        tmp = (message[i * 64: (i + 1) * 64])[::-1]
        h = self.__G_n(N, h, tmp)
        N = self.__AddModulo(N, N_512)
        Sigma = self.__AddModulo(Sigma, tmp)
        length_bits -= 512
        i += 1

    last = (message[i * 64: length])[::-1]

    if (len(last) == 0 and no_pad):
        pass
    else:
        h, N, Sigma = self.__Padding(last, N, h, Sigma)

    N_0 = [0] * 64
    h = self.__G_n(N_0, h, N)
    h = self.__G_n(N_0, h, Sigma)

    if out == 512:
        return h[::-1]
elif out == 256:
    return (h[0:32])[::-1]

def hash(self, str_message, out=512, no_pad=False):
    return list2str(self.GetHash(str2list(str_message), out, no_pad))

def H256(msg):
    S = Stribog()
    return S.hash(msg, out=256)

def H512(msg):
    S = Stribog()
    return S.hash(msg)

def num2le( s, n ) :
    res = ""
    for i in range(n):
        res += chr(s & 0xFF)
        s >>= 8
    return res

def le2num( s ) :
    res = 0
    for i in range(len(s) - 1, -1, -1):
        res = (res << 8) + ord(s[i])
    return res

def XGCD(a,b):
    """XGCD(a,b) returns a list of form [g,x,y], where g is GCD(a,b) and
    x,y satisfy the equation g = ax + by.""
    a1=1; b1=0; a2=0; b2=1; aneg=1; bneg=1; swap = False
    if(a < 0):
        a = -a; aneg=-1
    if(b < 0):
        b = -b; bneg=-1
    if(b > a):
        swap = True
        [a,b] = [b,a]
    while (1):
        quot = -(a / b)
        a = a % b
        a1 = a1 + quot*a2; b1 = b1 + quot*b2
        if(a == 0):
            if(swap):
                return [b, b2*bneg, a2*aneg]
            else:
                return [b, a2*aneg, b2*bneg]
        quot = -(b / a)
        b = b % a
        a2 = a2 + quot*a1; b2 = b2 + quot*b1
        if(b == 0):
            if(swap):
                return [a, b1*bneg, a1*aneg]
else:
    return [a, a1*aneg, b1*bneg]

def getMultByMask( elems, mask ):
    n = len( elems )
    r = 1
    for i in range( n ):
        if mask & 1:
            r *= elems[ n - 1 - i ]
        mask = mask >> 1
    return r

def subF(P, other, p):
    return (P - other) % p

def divF(P, other, p):
    return mulF(P, invF(other, p), p)

def addF(P, other, p):
    return (P + other) % p

def mulF(P, other, p):
    return (P * other) % p

def invF(R, p):
    assert (R != 0)
    return XGCD(R, p)[1] % p

def negF(R, p):
    return (-R) % p

def powF(R, m, p):
    assert R != None
    assert type(m) in (int, long)
    if m == 0:
        assert R != 0
        return 1
    elif m < 0:
        t = invF(R, p)
        return powF(t, (-m), p)
    else:
        i = m.bit_length() - 1
        r = 1
        while i > 0:
            if (m >> i) & 1:
                r = (r * R) % p
            r = (r * r) % p
i -= 1
if m & 1:
    r = (r * R) % p
return r

def add(Px, Py, Qx, Qy, p, a, b):
    if Qx == Qy == None:
        return [Px, Py]
    if Px == Py == None:
        return [Qx, Qy]
    if (Px == Qx) and (Py == negF(Qy, p)):
        return [None, None]
    if (Px == Qx) and (Py == Qy):
        assert Py != 0
        return duplicate(Px, Py, p, a)
    else:
        l = divF(subF(Qy, Py, p), subF(Qx, Px, p), p)
        resX = subF(powF(l, 2, p), Px, p)
        resY = subF(mulF(l, subF(Px, resX, p), p), Py, p)
        return [resX, resY]

def duplicate(Px, Py, p, a):
    if (Px == None) and (Py == None):
        return [None, None]
    if Py == 0:
        return [None, None]
    l = divF(addF(mulF(powF(Px, 2, p), 3, p), a, p), mulF(Py, 2, p), p)
    resX = subF(powF(l, 2, p), mulF(Px, 2, p), p)
    resY = subF(mulF(l, subF(Px, resX, p), p), Py, p)
    return [resX, resY]

def mul(Px, Py, s, p, a, b):
    assert type(s) in (int, long)
    assert Px != None and Py != None

    X = Px
    Y = Py

    i = s.bit_length() - 1
    resX = None
    resY = None
    while i > 0:
        if (s >> i) & 1:
            resX, resY = add(resX, resY, X, Y, p, a, b)
            resX, resY = duplicate(resX, resY, p, a)
        i -= 1
    if s & 1:
        resX, resY = add(resX, resY, X, Y, p, a, b)
return [resX, resY]

def Ord(Px, Py, m, q, p, a, b):
    assert Px != None and Py != None
    assert (m != None) and (q != None)
    assert mul(Px, Py, m, p, a, b) == [None, None]
    X = Px
    Y = Py
    r = m
    for mask in range(1 << len(q)):
        t = getMultByMask(q, mask)
        Rx, Ry = mul(X, Y, t, p, a, b)
        if (Rx == None) and (Ry == None):
            r = min(r, t)
    return r

def isQuadraticResidue( R, p ) :
    if R == 0:
        assert False
    temp = powF(R, ((p - 1) / 2), p)
    if temp == (p - 1):
        return False
    else:
        assert temp == 1
    return True

def getRandomQuadraticNonresidue(p):
    from random import randint
    r = (randint(2, p - 1)) % p
    while isQuadraticResidue(r, p):
        r = (randint(2, p - 1)) % p
    return r

def ModSqrt( R, p ) :
    assert R != None
    assert isQuadraticResidue(R, p)
    if p % 4 == 3:
        res = powF(R, (p + 1) / 4, p)
        if powF(res, 2, p) != R:
            res = None
        return [res, negF(res, p)]
    else:
        ainvF = invF(R, p)
        s = p - 1
        alpha = 0
        while (s % 2) == 0:
            alpha += 1
            s = s / 2
        b = powF(getRandomQuadraticNonresidue(p), s, p)
\[ r = \text{powF}(R, (s + 1) / 2, p) \]

\[ b_j = 1 \]

for k in range(0, alpha - 1):  # alpha >= 2 because p % 4 = 1
    d = 2 ** (alpha - k - 2)
    x = \text{powF}(\text{mulF}(\text{powF}(b_j, r, p), 2, p), \text{ainvF}, p, d, p)
    if x != 1:
        b_j = \text{mulF}(b_j, \text{powF}(b, (2 ** k), p), p)
    res = \text{mulF}(b_j, r, p)
return [res, \text{negF}(res, p)]

def generateQs(p, pByteSize, a, b, m, q, orderDivisors, Px, Py, N):
    assert pByteSize in (256 / 8, 512 / 8)
    PxBytes = \text{num2le}(Px, pByteSize)
    PyBytes = \text{num2le}(Py, pByteSize)
    Qs = []
    S = []
    Hash_src = []
    Hash_res = []
    co_factor = m / q
    seed = 0
    while len(Qs) != N:
        hashSrc = PxBytes + PyBytes + \text{num2le}(seed, 4)
        if pByteSize == (256 / 8):
            QxBytes = H256(hashSrc)
        else:
            QxBytes = H512(hashSrc)
        Qx = \text{le2num}(QxBytes) % p
        R = \text{addF}(\text{addF}(\text{powF}(Qx, 3, p), \text{mulF}(Qx, a, p), p), b, p)
        if (R == 0) or (not isQuadraticResidue(R, p)):
            seed += 1
            continue
        Qy_sqrt = \text{ModSqrt}(R, p)
        Qy = min(Qy_sqrt)
        if co_factor * Ord(Qx, Qy, m, orderDivisors, p, a, b) != m:
            seed += 1
            continue
        Qs += [(Qx, Qy)]
        S += [seed]
        Hash_src += [hashSrc]
        Hash_res += [QxBytes]
        seed += 1
    return Qs, S, Hash_src, Hash_res

if __name__ == '__main__':
    for i, curve in enumerate(curvesParams):
        print "A.1." + str(i+1) + ". Curve " + curve["OID"]
if "3410-2012-256-paramSetA" in curve["OID"] or "3410-2012-512-paramSetC" in curve["OID"]:
    Q, S, Hash_src, Hash_res = generateQs(curve["p"],
    curve["n"],
    curve["a"],
    curve["b"],
    curve["m"],
    curve["q"],
    [ 2, 2, curve["q"]],
    curve["x"],
    curve["y"],
    1)
else:
    Q, S, Hash_src, Hash_res = generateQs(curve["p"],
    curve["n"],
    curve["a"],
    curve["b"],
    curve["m"],
    curve["q"],
    [curve["q"]],
    curve["x"],
    curve["y"],
    1)

j = 1
for q, s, hash_src, hash_res in zip(Q, S, Hash_src, Hash_res):
    print "Point Q_" + str(j)
    j += 1

    print "X=", hex(q[0])[:-1]
    print "Y=", hex(q[1])[:-1]

    print "SEED=","{0:#0{1}x}".format(s,6)
    print

Authors' Addresses

Stanislav Smyshlyaev (editor)
CRYPTO-PRO
18, Suschevsky val
Moscow, 127018
Russian Federation
Phone: +7 (495) 995-48-20
EMail: svs@cryptopro.ru

Evgeny Alekseev
CRYPTO-PRO
18, Suschevsky val
Moscow, 127018
Russian Federation
Phone: +7 (495) 995-48-20
EMail: alekseev@cryptopro.ru
Igor Oshkin
CRYPTO-PRO
18, Suschevsky val
Moscow, 127018
Russian Federation
Phone: +7 (495) 995-48-20
EMail: oshkin@cryptopro.ru

Vladimir Popov
CRYPTO-PRO
18, Suschevsky val
Moscow, 127018
Russian Federation
Phone: +7 (495) 995-48-20
EMail: vpopov@cryptopro.ru