The Security Evaluated Standardized Password-Authenticated Key Exchange (SESPAKE) Protocol

Abstract

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

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8133.

Copyright Notice

Copyright (c) 2017 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.
1. Introduction

This document describes the Security Evaluated Standardized Password-Authenticated Key Exchange (SESPAKE) protocol. The SESPAKE protocol provides password-authenticated key exchange for usage in systems for protection of sensitive information. The protocol is intended to be used to establish keys that are then used to organize a secure channel for protection of sensitive information. The security proofs of the protocol were made for situations involving an active adversary in the channel, including man-in-the-middle (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**: 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 at 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 (MAC), 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 [RFC8018]. The parameter len is considered equal to the minimum integer that is a multiple of 8 and satisfies the following condition:

\[
\text{len} \geq \text{floor}(\log_2(q)).
\]
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 \( n, n \geq 0; \) 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 \( b = (b_1, \ldots, b_n), \) where \( b_1, \ldots, b_n \) are elements of \( \{0, \ldots, 255\}; \)

\( || \) concatenation of byte strings \( A \) and \( C, \) i.e., if \( A \in B_{n1}, \) \( C \in B_{n2}, \) \( A = (a_1, a_2, \ldots, a_{n1}) \) and \( C = (c_1, c_2, \ldots, c_{n2}), \) then \( A || C = (a_1, a_2, \ldots, a_{n1}, c_1, c_2, \ldots, c_{n2}) \) is an element of \( B_{(n1 + n2)}; \)

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

\( \text{bytes}_n(X) \) the byte string \( A \) in \( B_n, \) such that \( \text{int}(A) = X, \) where \( X \) is an integer and \( 0 \leq X < 256^n; \)

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

4. Protocol Description

The main point of the SESPAKE protocol is that parties sharing a weak key (a password) generate a strong common key. An active adversary who has 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 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, counters that indicate the number of failed connections were introduced in the SESPAKE protocol. There is also a special technique for small-order point processing and a mechanism that provides protection against reflection attacks 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, a pseudorandom function (PRF) used in the PBKDF2 function, etc.), which is a byte string of an arbitrary length;
- the point \( P \), which 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 the protocol instance.

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

The following protocol parameters are used by subject A:

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 the 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 \( C_{1A} \) counter, which 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, which 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, which 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 subject A (OPTIONAL), which is a byte string of an arbitrary length.

The following protocol parameters are used by subject B:

1. The values ind and salt, where ind is in $\{1, ..., N\}$ and salt is in $\{1, ..., 2^{128}-1\}$.

2. The point $Q_{PW}$, satisfying the following equation:

   $$Q_{PW} = \text{int}(F(PW, salt, 2000)) \cdot Q_{ind}.$$

   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 $\{Q_1, Q_2, ..., Q_N\}$.

3. The ID_ALG identifier.

4. The $C_1^B$ counter, which 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, which 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, which 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 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>$Q_PW^A = \text{int}(F(PW, salt, 2000)) \cdot Q_{ind}$</td>
<td>$Q_B = u_1 + Q_PW$</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 $u_1$ not in $E$ ==&gt; quit</td>
</tr>
<tr>
<td>$u_1 --&gt;$</td>
<td>$z_B = 0$</td>
</tr>
<tr>
<td>$u_1$ --&gt; $u_1$ not in $E$ ==&gt; quit</td>
<td>$Q_B = u_1 + Q_PW$</td>
</tr>
<tr>
<td>if $u_2$ not in $E$ ==&gt; quit</td>
<td>if $m/q \cdot Q_B = 0$ ==&gt; $Q_B = beta \cdot P$, $z_B = 1$</td>
</tr>
<tr>
<td>$Q_A = u_2 - Q_PW^A$</td>
<td>$K_B = \text{HASH}(\text{BYTES}({ m/q \cdot beta \cdot (\text{mod} q) } \cdot Q_B ))$</td>
</tr>
<tr>
<td>if $m/q \cdot Q_A = 0$ ==&gt; $Q_A = alpha \cdot P$, $z_A = 1$</td>
<td>$u_2 = beta \cdot P + Q_PW$</td>
</tr>
<tr>
<td>$K_A = \text{HASH}(\text{BYTES}({ m/q \cdot alpha \cdot (\text{mod} q) } \cdot Q_A ))$</td>
<td></td>
</tr>
</tbody>
</table>

The full description of the protocol consists of the following steps:

1. If any of the counters C_1^A, C_2^A, or C_3^A is equal to 0, A finishes the protocol with an informational error regarding 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, and 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, or C_3^B is equal to 0, B finishes the protocol with an informational error regarding 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, and 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 subsequent 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 subsequent calculations are done by A in this elliptic curve group.

7. A calculates the point $Q_\text{PW}^A = \text{int}(F(\text{PW}, \text{salt}, 2000)) \cdot Q_\text{ind}$.  

8. A chooses randomly (according to the uniform distribution) the value alpha; alpha is in $\{1, \ldots, q-1\}$; then A assigns $z_A = 0$.

9. A sends the value $u_1 = \alpha P - Q_\text{PW}^A$ to B.

10. After receiving $u_1$, B checks to see if $u_1$ is in $E$. If it is not, B finishes with an error and considers the authentication process unsuccessful.

11. B calculates $Q_B = u_1 + Q_\text{PW}$, assigns $z_B = 0$, and chooses randomly (according to the uniform distribution) the value beta; beta is in $\{1, \ldots, q-1\}$.

12. If $m/q \cdot Q_B = O$, B assigns $Q_B = \beta P$ and $z_B = 1$.

13. B calculates $K_B = \text{HASH}(\text{BYTES}((m/q \cdot \text{beta} \cdot (\text{mod } q)) \cdot Q_B))$.

14. B sends the value $u_2 = \beta P + Q_\text{PW}$ to A.

15. After receiving $u_2$, A checks to see if $u_2$ is in $E$. If it is not, A finishes with an error and considers the authentication process unsuccessful.

16. A calculates $Q_A = u_2 - Q_\text{PW}^A$.

17. If $m/q \cdot Q_A = O$, then A assigns $Q_A = \alpha P$ and $z_A = 1$.

18. A calculates $K_A = \text{HASH}(\text{BYTES}((m/q \cdot \text{alpha} \cdot \text{mod } q) \cdot Q_A))$.

19. A calculates $U_1 = \text{BYTES}(u_1)$, $U_2 = \text{BYTES}(u_2)$.

20. A calculates $\text{MAC}_A = \text{HMAC}(K_A, 0x01 || \text{ID}_A || \text{ind} || \text{salt} || U_1 || U_2 || \text{ID_ALG} (\text{OPTIONAL}) || \text{DATA}_A)$, 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 to see if the values $\text{MAC}_A$ and $\text{HMAC}(K_B, 0x01 || \text{ID}_A \text{ || ind || salt || U}_1 \text{ || U}_2 || \text{ID}_A \text{ || ID}_A \text{ || } \text{DATA}_A)$ are equal. If they are not, it finishes with an error and considers the authentication process unsuccessful.

24. If $z_B = 1$, B finishes with an error and considers the authentication process unsuccessful.

25. B sets the value of $C_1^B$ to CLim and increments $C_2^B$ by 1.

26. B calculates $\text{MAC}_B = \text{HMAC}(K_B, 0x02 || \text{ID}_B \text{ || ind || salt || U}_1 \text{ || U}_2 || \text{ID}_A \text{ || ID}_A \text{ || DATA}_A \text{ || DATA}_B)$, where $\text{DATA}_B$ is an OPTIONAL string that is authenticated with $\text{MAC}_B$ (if it is not used, then $\text{DATA}_B$ is considered to be of zero length).

27. B sends $\text{DATA}_B$, $\text{MAC}_B$ to A.

28. A checks to see if the values $\text{MAC}_B$ and $\text{HMAC}(K_A, 0x02 || \text{ID}_B \text{ || ind || salt || U}_1 \text{ || U}_2 || \text{ID}_A \text{ || ID}_A \text{ || DATA}_A \text{ || DATA}_B)$ are equal. If they are not, it finishes with an error and considers the authentication process unsuccessful.

29. If $z_A = 1$, A finishes with an error and considers the authentication process unsuccessful.

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

After the procedure finishes successfully, subjects A and B are mutually authenticated, and each subject has an explicitly authenticated value of $K = K_A = K_B$.

Notes:

1. In cases where 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 to see if 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 (a zero-length string is allowed) defined by a specific implementation.
2. The ind, ID_A, ID_B, and salt parameters can be agreed upon in advance. If some parameter is agreed upon in advance, it is possible not to send it during a corresponding step. Nevertheless, all parameters MUST be used as corresponding inputs to the HMAC function during Steps 20, 23, 26, and 28.

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

4. It is RECOMMENDED that the ID_ALG parameter be used in HMAC during Steps 20, 23, 26, and 28.

5. Continuation of protocol interaction in a case where any of the counters C_1^A or C_1^B is equal to zero MAY be done without changing the 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 period.

6. Continuation of protocol interaction in a case where any of the counters C_2^A, C_3^A, C_2^B, or C_3^B is equal to zero MUST be done only after changing the password.

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

8. The use of several Q points can reinforce the independence of the data streams when working with several applications -- for example, when 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 a possible algorithm for the 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 = int(HASH(BYTES(P) || bytes_s(SEED))) mod p.

3. Check to see if 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{r1, r2}, where r1, r2 from {0,1,...,p-1} are such that r1 != r2 and r1^2 = r2^2 = R mod p for R = X^3 + aX + b.

5. Check to see if the following relations hold for the point Q = (X, Y): Q != 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; otherwise, finish.

With the defined algorithm for any elliptic curve E, point sets \{Q_1,...,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 knowledge could be used to implement an attack against the protocol with an exhaustive search for the password.
6. Security Considerations

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

It is RECOMMENDED that the HASH function satisfy the following condition:

- hashlen ≤ log_2(q) + 4, where hashlen is the length of the HASH function output.

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

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

Using N = 1 is RECOMMENDED.

Note: The specific adversary models for the protocol discussed in this document can be found in [SESPAKE-SECURITY], which contains the security proofs.

7. IANA Considerations

This document does not require any IANA actions.
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 for $N = 1$ and the GOST R 34.11-2012 hash function (see [RFC6986]). If $2^{254} < q < 2^{256}$, the GOST R 34.11-2012 hash function with 256-bit output is used, and if $2^{508} < q < 2^{512}$, the GOST R 34.11-2012 hash function with 512-bit output is used.

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 also by a pair of $(U, V)$ coordinates in the twisted Edwards form in accordance with [RFC7836] for the curves that have equivalent representations in this form. There is a SEED value for each point, by which it was generated.

id-GostR3410-2001-CryptoPro-A-ParamSet, id-GostR3410-2001-CryptoPro-B-ParamSet, etc. are defined in [RFC4357]. id-tc26-gost-3410-2012-512-paramSetA, id-tc26-gost-3410-2012-512-paramSetB, etc. are defined in [RFC7836].


Point $Q_1$
X = $0xa69d51caf1a309fa9e9b66187759b0174c274e080356f23cfcbfe84d396ad7bb$
Y = $0x5d26f29ecc2e9ac0404dcf7986fa55fe94986362170f54b9616426a659786dac$
SEED = $0x0001$

A.1.2. Curve id-GostR3410-2001-CryptoPro-B-ParamSet

Point $Q_1$
X = $0x3d715a874a4b17cb3b517893a9794a2b36c89d2ff6693f01ee4cc27e7f9e399$
Y = $0x1c5a641fcf7ce7e87c9df8ce838f3db3096eace2fad158384535365f4fe7fe$
SEED = $0x0000$

A.1.3. Curve id-GostR3410-2001-CryptoPro-C-ParamSet

Point $Q_1$
X = $0xe36383e43bb6ca2917167d71b7b5dd3d6d462b43d7c64282ae67dfbec2559d$
Y = $0x137478a9f721c73932ea06b45cf72e37eb78a63f29a542e563c614650c8b6399$
SEED = $0x0006$

Smyshlyaev, et al.                       Informational
A.1.4. Curve id-tc26-gost-3410-2012-512-paramSetA

Point Q_1
X = 0xa7f8833a23795327478871b5c5e88ae4be91126c64b4b327289bea62559425
d18198f133f400874328b220c74497cd240586cb249e158532cb8090776d61c
Y = 0x728f0c4a73b48da41ce928358fad26b47a6e094e9362bae82559f983cddd4ec3a
4676bd3707edeadf4c85e99695c64c241ec622be87dc0c8f751f4367f723c5
SEED = 0x0001

A.1.5. Curve id-tc26-gost-3410-2012-512-paramSetB

Point Q_1
X = 0x7e1fae8285e035bec244be4f2d0e5ebf436633cf50e55231dea9c9cf214c8c33
df54d305e92971fa0b4c07e00d87bdb5720eb66e49079285aaf12e0171149
Y = 0x2cc89998b87fd443805ba0d58a196592db20ab161558ff2f4e77a5752d5209
53967ae621afdeae89bb77c83a2528ef6f6e02f68bda4679d7f2704947db408
SEED = 0x0000

A.1.6. Curve id-tc26-gost-3410-2012-256-paramSetA

Point Q_1
X = 0xb51adf93a40ab15792164fad3352f95b66369eb24aef5efae32829320363350e
Y = 0x74a358cc08593612f5955d249c96af87e8b6bb6d8bd22b4e91046650d822be8
U = 0xebe977affe00df888b80114b8de430ac2b34564e4240af74278e7305bc48eaa
V = 0x828f2df8f06612b4f4ea4da72ca509c0f76dd37df424ea22bfa6f4f65748c1e4
SEED = 0x0001

A.1.7. Curve id-tc26-gost-3410-2012-512-paramSetC

Point Q_1
X = 0x489c91784e02e98f19a803abca319917f37689e5a18965251ce2ff4e8d8b298f
5ba7470f9e0e713487f96f4a8397b3d09a270c9d367eb5e6e8561adeeb51581d
Y = 0x684ea885aca64efa1b3f3e36c0852a3eb3d8011b0ef18e205f87028d65e5db
2c144a0ddc71276542fbdf72c2a2a4f34af4939da66d9a60793c70a48c94e16f18
U = 0x3a3496f97e6b3849a4fa7db60fd93858bde89958e4beed05a6b3214261b37c
9d9a560076e7ea59714828b18ffbef996f9c98bfc2dc923ebead36a0dace88
V = 0x52d8d84c8bf0ad6c5f7b3973e32a668daafl1ed092efl38dbae6203b2ccdec561
47464d35f3e4b727b2480eb143074712c76550c7a5df3ea26f70059480dcb50
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 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.


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 = 0xA69D51CAF1A309FA9E9B66187759B0174C274E080356F23CFCBFE84D396AD7BB
    Y = 0x5D26F29ECC2E9AC0404DCF7986FA55FE94986362170F54B9616426A659786DAC

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 = 0x59495655D1E7C7424C622485F575CCF121F3122D274101E8AB734CC9C9A9B45E
    Y = 0x48D1C311D33C9B701F3B03618562A4A07A044E3AF31E3999E67B487778B53C62

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

$alpha=0x1F2538097D5A031FA68BBB43C84D12B3DE47B7061C0D5E24999E0C873CDBA6B3$

$alpha*P$:
    X = 0xBBC77CF42DC1E62D06227935379B4AA4D14FEA4F565DDF4CB4FA4D31579F9676
    Y = 0x8E16604A4AFDF28246684D4996274781F6CB80ABBBAA1414C1513EC988509DABF

$u_1$:
    X = 0x204F564383B2A76081B907F3FCA8795E806BE2C2ED228730B5B9E37074229E8D
    Y = 0xE84F9E442C61DDE37B601A7F37ECA11C56183FA071DFA9320EDE3E7521F9D41
When processing \( u_1 \), calculating the \( K_B \) key, and calculating \( u_2 \) on subject \( B \), the parameters \( \beta, \text{src}, K_B = \text{HASH(src)}, \beta^*P, \) and \( u_2 \) take the following values:

\[
\begin{align*}
\beta &= 0xDC497D9EF6324912FD367840EE509A2032AEDB1C0A890D133B45F596FCCBD45D \\
\text{src}: &\quad 2E 01 A3 D8 4F DB 7E 94 7B B8 92 9B E9 36 3D F5 \\
&\quad F7 25 D6 40 1A A5 59 D4 1A 67 24 F8 D5 F1 8E 2C \\
&\quad A0 DB A9 31 05 CD DA F4 BF AE A3 90 6F DD 71 9D \\
&\quad BE B2 97 B6 A1 7F 4F BD 96 DC C7 23 EA 34 72 A9 \\
K_B: &\quad 1A 62 65 54 92 1D C2 E9 2B 4D D8 D6 7D BE 5A 56 \\
&\quad 62 E5 62 99 37 3F 06 79 95 35 AD 26 09 4E CA A3 \\
\beta^*P: &\quad X = 0x6097341C1BE388E83E7CA2DF47FAB86E2271FD942E5B72EB2409E49F742BC29 \\
&\quad Y = 0xC81AA48BDB4CA6FA0EF18B9788AE25FE30857AA681B3942217F9FED151BAB7D0 \\
u_2: &\quad X = 0xDC137A2F1D4A35AEBC0ECBF6D3486DEF8480BFDC752A86DD4F207D7D1910E22D \\
&\quad Y = 0x7532F0CE99DCC772A4D7861DAE57C138F07AE304A727907FB0AAFDB624ED572
\end{align*}
\]

When processing \( u_2 \) and calculating the key on subject \( A \), the \( K_A \) key takes the following values:

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

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

\[
\text{MAC}_A: \\
&\quad 23 7A 03 C3 5F 49 17 CE 86 B3 58 94 45 F1 1E 1A \\
&\quad 6F 10 8B 2F DD 0A A9 E8 10 66 4B 25 59 60 B5 79
\]

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

\[
\text{MAC}_B: \\
&\quad 9E E0 E8 73 3B 06 98 50 80 4D 97 98 73 1D CD 1C \\
&\quad FF E8 7A 3B 15 1F 0A E8 3E A9 6A FB 4F FC 31 E4
\]
A.2.2. Curve id-GostR3410-2001-CryptoPro-B-ParamSet

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

\[ N = 1 \]
\[ \text{ind} = 1 \]
\[ \text{ID}_A: \quad 00\ 00\ 00\ 00 \]
\[ \text{ID}_B: \quad 00\ 00\ 00\ 00 \]
\[ \text{PW:} \quad 31\ 32\ 33\ 34\ 35\ 36\ ('123456') \]
\[ \text{salt:} \quad 29\ 23\ BE\ 84\ E1\ 6C\ D6\ AE\ 52\ 90\ 49\ F1\ F1\ BB\ E9\ EB \]
\[ Q_{\text{ind}}: \]
\[ X = 0x3D715A874A4B17CB3B517893A9794A2B36C89D2F693F01EE4CC27E7F49E399 \]
\[ Y = 0x1C5A641FCE7E87CDF8CEA38F3DB3096EACE2FAD158384B53953365F4FE7FE \]

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

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

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

\[ X = 0x6DC2AE26BC691FCA5A73D9C452790D15E34BA5404D92955B914C8D2662ABB985 \]
\[ Y = 0x3B02AAA9DD65AEC335C35C3E2F13154BBAC059F66B088306747553EDF65DB77 \]

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

\[ \alpha = 0x499D72B90299CAB0DA1F88BE19D9122F622A13B32B730C46BD0664044F2144FAD \]
\[ \alpha P: \]
\[ X = 0x61D6F916DB717222D74877F179F7EBEF7CD4D24D8C1F523C048E34A1DF30F8DD \]
\[ Y = 0x3EC48863049CFCEF662904082E78503F4973A4E105E2F1B18C69A5E7FB20900 \]
\[ u_1: \]
\[ X = 0x21F5437AF33D2A1171A070226B4AE82D3765C00EBFF1EC16FE158EBC50C63AB1 \]
\[ Y = 0x5C9553B5D11AAECE738AD9A9F8CB4C100AD4FA5E089D3CBCCEA8C0172EB7ECC \]
When processing u_1, calculating the K_B key, and calculating u_2 on subject B, the parameters beta, src, K_B = HASH(src), beta*P, and u_2 take the following values:

\[
\begin{align*}
\beta &= 0x0F69FF614957EF83668EDC2D7ED614BE76F7B253DB23C5CC9C52BF7DF8F4669D \\
\text{src:} &= \text{50 14 0A 5D ED 33 43 EF C8 25 7B 79 E6 46 D9 F0} \\
&\quad \text{DF 43 82 8C 04 91 9B D4 60 C9 7A D1 4B A3 A8 6B} \\
&\quad \text{00 C4 06 B5 74 4D 8E B1 49 DC 8E 7F C8 40 64 D8} \\
&\quad \text{53 20 25 3E 57 A9 B6 B1 3D 0D 38 FE A8 EE 5E 0A} \\
K_B: &= \text{A6 26 DE 01 B1 68 0F F7 51 30 09 12 2B CE E1 89} \\
&\quad \text{68 83 39 4F 96 03 01 72 45 5C 9A E0 60 CC E4 4A} \\
\beta*P: &= \text{X = 0x33BC6F7E9C0BA10CFB2B72546C327171295508EA97F8C8BA9F890F2478AB4D6C} \\
&\quad \text{Y = 0x75D57B396C396F492F057E9222CCC686437A2AAD464E452EF426FC8EEED1A4A6} \\
u_2: &= \text{X = 0x089DDEE718EE8A224A7F37E22CFFD731C25FCBF58860364EE322412CDCEF99AC} \\
&\quad \text{Y = 0x0ECE03D4E395A6354C571871BEF425A532D5D463B0F8FD427F91A43E20CDA55C}
\end{align*}
\]

When processing u_2 and calculating the key on subject A, the K_A key takes the following values:

\[
\begin{align*}
K_A: &= \text{A6 26 DE 01 B1 68 0F F7 51 30 09 12 2B CE E1 89} \\
&\quad \text{68 83 39 4F 96 03 01 72 45 5C 9A E0 60 CC E4 4A} \\
The \text{message MAC_A = HMAC(K_A, 0x01 || ID_A || ind || salt || u_1 || u_2)} \\
&\text{from subject A takes the following values:}
\end{align*}
\]

\[
\begin{align*}
\text{MAC_A:} &= \text{B9 1F 43 90 2A FA 90 D3 E5 C6 91 CB DC 43 8A 1E} \\
&\quad \text{BF 54 7F 4C 2C B4 14 43 CC 38 79 7B E2 47 A7 D0}
\end{align*}
\]

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

\[
\begin{align*}
\text{MAC_B:} &= \text{79 D5 54 83 FD 99 B1 2B CC A5 ED C6 BB E1 D7 B9} \\
&\quad \text{15 CE 04 51 B0 89 1E 77 5D 4A 61 CB 16 E3 3F CC}
\end{align*}
\]

Smyshlyaev, et al.

Informational

[Page 21]
A.2.3. Curve id-GostR3410-2001-CryptoPro-C-ParamSet

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

\[
\begin{align*}
N &= 1 \\
\text{ind} &= 1 \\
\text{ID}_A &: \quad 00 00 00 00 \\
\text{ID}_B &: \quad 00 00 00 00 \\
\text{PW} &: \quad 31 32 33 34 35 36 ('123456') \\
\text{salt} &: \quad 29 23 84 6E 5C 49 01 2F 1F 1B 1E 1E 1E 1E 1E 1E \\
\text{Q}\_\text{ind} &: \quad X = 0x1E36383E43BB6CFA2917167D71B7B5DD3D6D462B43D7C64282AE67DFBEC2559D \\
& \quad Y = 0x137478A9F721C73932EA06B45CF72E37EB78A63F29A542E563C614650C8B6399 \\
\text{F}(\text{PW}, \text{salt}, 2000) & \quad \text{takes the following values:} \\
\text{F}(\text{PW}, \text{salt}, 2000) &: \quad BD 04 67 3F 71 49 B1 8E 98 15 5B D1 E2 72 4E 71 \\
& \quad D0 09 9A A2 51 74 F7 92 D3 32 6C 6F 18 12 70 67 \\
\text{Q}\_\text{PW} &: \quad \text{X} = 0x945821DAF91E158B839939630655A3B21FF3E146D27041E86C05650EB3B46B59 \\
& \quad \text{Y} = 0x3A0C2816AC97421FA0E879605F17F0C9C3EB734CFF196937F6284438D70BDC48 \\
\end{align*}
\]

The coordinates of the point Q_PW are:

\[
\begin{align*}
\text{X} &= 0x945821DAF91E158B839939630655A3B21FF3E146D27041E86C05650EB3B46B59 \\
\text{Y} &= 0x3A0C2816AC97421FA0E879605F17F0C9C3EB734CFF196937F6284438D70BDC48 \\
\end{align*}
\]

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

\[
\begin{align*}
\text{alpha} &= 0x3A54AC3F19AD9D0B1EAC8ACDCEA70E581F1DAC33D13FEAFD81E762378639C1A8 \\
\text{alpha}\_P &: \quad \text{X} = 0x96B7F09C94D297C257A7DA48364C0076E59E48D221CBA604AE111CA3933B446A \\
& \quad \text{Y} = 0x54E4953D86B77ECCEB578500931E822300F7E091F79592CA202A02076234A6 \\
\text{u}_1 &: \quad \text{X} = 0x81BBD6FCA464D2E2404A66D786CE4A777E739A89AE868C2DCA29D53273B75387 \\
& \quad \text{Y} = 0x6B6DBD922EA7E60998F8B230AB6EF07AD2EC86B2BF66391D82A30612EADD411 \\
\end{align*}
\]
When processing u_1, calculating the K_B key, and calculating u_2 on subject B, the parameters beta, src, K_B = HASH(src), beta*P, and u_2 take the following values:

\[
\begin{align*}
\text{beta} &= 0x448781782BF7C0E52A1DD9E6758FD3482D90D3CFCCF42232CF357E59A4D49FD4 \\
\text{src} &= \text{16 A1 2D 88 54 7E 1C 90 06 BA A0 08 E8 CB EC C9} \\
&\quad \text{D1 68 91 ED C8 36 CF B7 5F 8E B9 56 FA 76 11 94} \\
&\quad \text{D2 8E 25 DA D3 81 8D 16 3C 49 4F 05 9A BC 70 A5} \\
&\quad \text{A1 B8 8A 7F 80 A2 EE 35 49 30 18 46 54 2C 47 OB} \\
\text{K_B} &= \text{BE 7E 7E 47 B4 11 16 F2 C7 7E 3B 8F CE 40 30 72} \\
&\quad \text{CA 82 45 0D 65 DE FC 71 A9 56 49 E4 DE EA EC EE} \\
\text{beta*P} &= \text{X = 0x4B9C0AB55A938121F282F48A2CC4396EB16E7E0068B495B0C1DD4667786A3EB7} \\
&\quad \text{Y = 0x223460AA8E09383E9DF9844C5A0F27664B4738E5B30128A171B69A77D9509B96} \\
\text{u_2} &= \text{X = 0x2ED9B903254003A672E89EBEB9C9E31503726AD124BB5FC0A726EE0E6FCCE323E} \\
&\quad \text{Y = 0x4CF5E1042190120391EC8DB62FE25E926EC60FB0B78B242199839C295FCD022} \\
\end{align*}
\]

When processing u_2 and calculating the key on subject A, the K_A key takes the following values:

\[
\begin{align*}
\text{K_A} &= \text{BE 7E 7E 47 B4 11 16 F2 C7 7E 3B 8F CE 40 30 72} \\
&\quad \text{CA 82 45 0D 65 DE FC 71 A9 56 49 E4 DE EA EC EE} \\
\end{align*}
\]

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

\[
\begin{align*}
\text{MAC_A} &= \text{D3 B4 1A E2 C9 43 11 36 06 3E 6D 08 A6 1B E9 63} \\
&\quad \text{BD 5E D6 A1 FF F9 37 FA 8B 09 0A 98 E1 62 BF ED} \\
\end{align*}
\]

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

\[
\begin{align*}
\text{MAC_B} &= \text{D6 B3 9A 44 99 BE D3 E0 4F AC F9 55 50 2D 16 B2} \\
&\quad \text{CB 67 4A 20 5F AC 3C D8 3D 54 EC 2F D5 FC E2 58} \\
\end{align*}
\]
A.2.4. Curve id-tc26-gost-3410-2012-512-paramSetA

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 = 0x2A17F8833A32795327478871B5C5E88AEFB91126C64B4B8327289BEA62559425
    D18198F133F400874328B220C74497CD2405866CB249E158532CB8090776CD61C
Y = 0x728F0C4A73B48DA41CE928358FA47B7A6E094E9362BAE82559F83CD4C4E3A
    4676BD3707EDEAF4CD85E99695C64C241EDC622BE87DC0CF87F51F4367F723C5

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

F(PW, salt, 2000):
BD 04 67 3F 71 49 B1 9E 98 15 5B D1 E2 72 4E 71
D0 09 9A A2 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 0D 69 5D FD 9F 5F 0F D5 AB A7 02

The coordinates of the point Q_PW are:

X = 0x0C0AB53D0E0A9C607CAD758F558915A0A7DC5DC87B45E9A58FDDF30EC3385960
    283E030CD322D9E46B070637785FD492CD711F46807A24C40AF9A42C8E2D740
Y = 0xDF93A8012B86D3A3D4F8A4D487DA15FC739EB31B20B3B0E8C8C032A4F8072C63
    37CF7D5B404719E5B4407C41D9A3216A08CA69C271484E9ED72B8AAA52E28B8B
During the calculation of $u_1$ on subject A, the parameter $\alpha$, the point $\alpha*P$, and $u_1$ take the following values:

$$\alpha=0x3CE54325DB52FE798824AEAD11BB16FA766857D04A4AF7D468672F16D90E7396$$  
$$046A46F815693E85B1CE5464DA9270181F8233B07150577BEE8D61D40050F0E$$

$$\alpha*P:$$

$$\begin{align*}
X &= 0xB93093EB0FCC463239B7DF276E9E592FCFC9B635504EA4531655D76A0A3078E \\
Y &= 0x809770BD910EA30BD2FA89736E91DC3181D29B3112807FEDC371E9F69466 \\
   & \quad F497DC64DD5B1F4005C95D61D400505F0E
\end{align*}$$

$$u_1:$$

$$\begin{align*}
X &= 0xE7510A9EDD37B869566C81052E2515E1563FDFE79F1D782D62000F33C3CC2764D \\
Y &= 0x809770BD910EA30BD2FA89736E91DC3181D29B3112807FEDC371E9F69466 \\
   & \quad F497DC64DD5B1F4005C95D61D400505F0E
\end{align*}$$

When processing $u_1$, calculating the $K_B$ key, and calculating $u_2$ on subject B, the parameters $\beta$, src, $K_B = HASH(src)$, $\beta*P$, and $u_2$ take the following values:

$$\beta=0xB5C286A79AA8E97EC0E1959C1D15F12F8C97870BA9D68CC12811A56A3BB$$  
$$1440610825796A49D468C6CD9C202D276598A27973D5960C5F0BCE28D8D345F4$$

$$\text{src:}$$

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

$$K_B:$$

$$\begin{align*}
53 & 24 DE F8 48 B6 63 CC 26 42 2F 5E 45 EE C3 4C \\
51 & D2 43 61 B1 65 60 CA 58 A3 D2 28 45 86 CB 7A \\
\end{align*}$$

$$\text{beta*P:}$$

$$\begin{align*}
X &= 0x238B3664E440452A99FA6B93D9FD7DA0CB83C32D3C1E3CFE5DF5C3EB0F9DB91 \\
Y &= 0x8982B1C374C33C71F887D20C7AF \\
\end{align*}$$

$$u_2:$$

$$\begin{align*}
X &= 0xC3384126216E81372001E77CIF9E7C547F89223CF7BB865C2472EC18BE0C79A \\
Y &= 0x8B520D903AFA257E8A54EC90CBADBAF4FEDD2C2D868C82FF04FCCB89EF6F38E56 \\
   & \quad F6B9F9472D477414DA7E36F538ED223D2E2EE02FAE1A20A98C5A9FCF03B6F3D0
\end{align*}$$
When processing u_2 and calculating the key on subject A, the K_A key takes the following values:

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 subject A takes the following values:

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 subject B takes the following values:

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
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 = 0x7E1FAE8285E035BEC244BEF2D0E5EBF436633CF50E55231DEA9C9CF21D4C8C33
DF85D4305DE92971FOA4B4C07E0D87BDBC720EB66E49079285AAAF12E0171149
Y = 0x2CC89998B875D4463B05BA0D858A196592DB20AB161558FF2F4EF7A85725D209
53967AE621AFDEAE89BB77C83A2528EF6FCE02F68BDA4679D7F2704947DBC408
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
- 1C 62 13 E3 93 0E FD DA 26 45 17 92 C6 20 81 22
- EE 60 D2 00 52 0D 69 5F FD 9F 5F 0F D5 AB A7 02

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

- $X = 0x7D03E65B8050D1E12CBB601A17B9273B0E728F5021CD47C8A4DD822E4627BA5F$
- $9C692686A2CDDA9A065509866B42EDEDEC4A118409604AD549F87A60AFA621161$
- $Y = 0x16037DAD45421EC50B00D50BDC6AC3B85348BC1D3A2F85DB27C3373580PEF87C$
- $2C743B7ED30F22BE22958044E716F93A61CA3213A361A2797A16A3AE62957377$

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

- $alpha=0x715E893FA639BF341296E0623E6D29DADF26B163C278767A7982A989462A3863$
- $FE12AEF8BD403D59C4DC4720570D4163DB0805C7C10C4E818F9CB785B04B9997$
- $alpha*P:$
  - $X = 0x10C479EAA1C04D3C2C02B0576A9C42D6226FF033C1191436777F66916030D87D$
  - $02FB93738ED769D07619F4C7C1F3C4DB5E5DF49E2186D6FAE2E2B5767602B9$
  - $Y = 0x039F604419419404E707F26D59D979136A831CCE43E1C5F060D1DDF83F39D0CA3D$
  - $52FBD943BF04DCCED1AA2CE8F5EBD7487ACDE2F239C07D015084D796784F35436$
- $u_1:$
  - $X = 0x45C05CE8290762F2470B719B4306D62B2911CEBL44F7F72EF11D10498C7E921$
  - $FF163F872044B4E7332AD8CBE3C12117820F53A60762315BCEB5BC6DA5CF1E0$
  - $Y = 0x5BE483E382D0F5F0748C4F6A504D99E62755B5ACC9554EC4A5B2093E121A2DD$
  - $5C6066BC9EDE39373BA19899208BB419E38B39BBDEDEB0B095CAEAA984D02E$
When processing \( u_1 \), calculating the \( K_B \) key, and calculating \( u_2 \) on subject B, the parameters \( \beta, \text{src}, K_B = \text{HASH(src)}, \beta*P \), and \( u_2 \) take the following values:

\[
\beta = 0x30FA8C2B4146C2DBBE82BED04D73788778C06753BD0A0FF71EBF2BEFE8DA8F3
\DC0836468E2CE7C5961281B6505140F8407413F03C2CB1D201EA1286CE30E6D
\]

\[
\text{src}:
3F 04 02 E4 0A 9D 59 63 20 5B CD F4 FD 89 77 91
9B BA F4 80 F8 E4 FB D1 25 5A EC E6 ED 57 26 4B
D0 A2 87 98 4F 59 D1 02 04 B5 F4 5E 4D 77 F3 CF
8A 63 B3 1B EB 2D F5 9F 8A F7 3C 20 9C CA 8B 50
B4 18 D8 01 E4 90 AE 13 3F 04 F4 F3 F4 D8 FE 8E
19 64 6A 1B AF 44 D2 36 FC C2 1B 7F 4D 8F C6 A1
E2 9D 6B 69 AC CE ED 4E 62 AB B2 0D AD 7B A5 F4
FE B0 ED 83 8E D9 1E 92 12 AB A3 89 71 4E 56 0C
\]

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

\[
\beta*P:
X = 0x34C0149E7BB91AE377B02573FCC48AF7BFB7B16DEB8F9CE870F384688E3241A3
A868588CC0EF4364CA67DE360CD82485C202ADC76F895D5DF673B1788E67
Y = 0x608E944929BD643569ED51871453F13333A1EAF82B2FE81BE8100E775F13D
D9925BD317B63BFAF05024D4A738852332B64501195C12BFE789E34F23DAFC5
\]

\[
u_2:
X = 0x0535F95463444C4594B5A2E4B35760491C670925060B4BEBC97DE3A3076D1A5
81F89026E04282B040925D925020124ACA4B2713569B6C3916A6F3344B840AD
Y = 0x40E6C2E55AEC31E7BCB6EA0242857FC6DFFB5409803EDF4CA20141F72CC3C7988
706E076765F4F004340E5294A7F8E53BA59CB67502F0044558C854A7D63FE900
\]

When processing \( u_2 \) and calculating the key on subject A, the \( K_A \) key takes the following values:

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

The message \( \text{MAC}_A = \text{HMAC}(K_A, 0x01 || \text{ID}_A || \text{ind} || \text{salt} || u_1 || u_2) \) from subject A takes the following values:

\[
\text{MAC}_A:
DE 46 BB 4C 8C E0 8A 6E F3 B8 DF AC CC 1A 39 B0
8D 8C 27 B6 CB 0F CF 59 23 86 A6 48 F4 E5 BD 8C
\]
The message MAC_B = HMAC(K_B, 0x02 || ID_B || ind || salt || u_1 || u_2) from subject B takes the following values:

MAC_B:
EC B1 1D E2 06 1C 55 F1 D1 14 59 CB 51 CE 31 40
99 99 99 2F CA A1 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
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 = 0xB51ADF93A40AB15792164FAD3352F95B66369EB2A4EF5EFAE32829320363350E
Y = 0x74A358CC08593612F5955D249C96AFB7E8B0BB6D8BD2BBE491046650D822BE18

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 = 0xDBF99827078956812FA48C6E695DF589DEF1D18A2D4D35A96D75BF6854237629
Y = 0x9FDDD48BFBC57BEE1DA0CFF282884F284D471B388893C48F5ECB02FC18D67589

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

alpha=0x147B72F6684F8BD1B418A899F7DBECFAF5FCE60B13685BAA95328654A7F0707F
alpha*P:
X = 0x33FBAC14EAE58275A769417829C421BD9FA622B6F0247EF55BD60E6E6BC2888
Y = 0x22F2EBCF960A82E6CDB4042D3DDDA511B2FBA925383C2273D952EA2D406EAE46
u_1:
X = 0xE569AB544E3A13C41077DE97D659A1B7A13F61DDD808BB633A5621FE2583A2C43
Y = 0xA21A743A08F4D715661297ECD6F86553A808925BF34802BF7EC34C548A40B2C0
When processing $u_1$, calculating the $K_B$ key, and calculating $u_2$ on subject B, the parameters $beta$, $src$, $K_B = HASH(src)$, $beta*P$, and $u_2$ take the following values:

$beta=0x30D5CFADAA0E31B405E6734C03EC4C5DF0F02F4BA25C9A3B320EE6453567B4CB$

$src:$
A3 39 A0 B8 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 0D E4 03 E8 17 C6 20 4B
C1 91 65 B9 D1 00 2B 9F 10 88 A6 CD A6 EA CF 27

$beta*P:$
$X = 0x2B2D89FAB735433970564F28CF6A82EEF170F942A81D6B4CE5DEC0DDB9447512962874870E6F2849A96F$
$Y = 0x10EF6A82EEF170F942A81D6B4CE5DEC0DDB9447512962874870E6F2849A96F$

$u_2:$
$X = 0x190D2F283F7E861065DB53227D7FBDF429CEBF93791262CB29569BDF63C86CA4$
$Y = 0xB3F1715721E9221897CCDE046C9B843A8386DBF7818A112F15A02BC820AC8F6D$

When processing $u_2$ and calculating the key on subject A, the $K_A$ key takes the following values:

$K_A:$
7D F7 1A C3 27 ED 51 7D 0D E4 03 E8 17 C6 20 4B
C1 91 65 B9 D1 00 2B 9F 10 88 A6 CD A6 EA CF 27

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

$MAC_A:$
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 34 4C 6E 6B A3 17 7F

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

$MAC_B:$
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:

\[
\begin{align*}
N &= 1 \\
\text{ind} &= 1 \\
\text{ID}_A &= \begin{array}{cccc}
0 & 0 & 0 & 0 \\
\end{array} \\
\text{ID}_B &= \begin{array}{cccc}
0 & 0 & 0 & 0 \\
\end{array} \\
\text{PW} &= \begin{array}{cccc}
31 & 32 & 33 & 34 \quad 35 \quad 36 \quad ('123456') \\
\end{array} \\
\text{salt} &= \begin{array}{cccc}
29 & 23 & BE & 84 \quad E1 \quad 6C \quad D6 \quad AE \quad 52 \quad 90 \quad 49 \quad F1 \quad F1 \quad EB \\
\end{array} \\
\text{Q\_ind} &= \begin{array}{cccc}
X &= \text{0x489C91784E02E98F19A803ABCA319917F37689E5A18965251CE2FF4E8D8BB298F} \\
&= 5BA7470F9E0713487F964A89973BD09A270C9D367EB5E06561ADDEE51581D \\
Y &= \text{0x684EA885ACA64EAF1B3FEE3EC0852A3BE3BD8011B08F18E203FF87028D6E65DB} \\
&= 2C144A0DCC71276542BFD72CA2A43FA4F4939DA66D9A6793C704A8C94E16F18 \\
\end{array} \\
\end{align*}
\]

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

\[
\begin{align*}
\text{F(\text{PW}, \text{salt}, 2000)} &= \begin{array}{cccc}
BD & 04 & 67 & 3F \quad 71 \quad 49 \quad B1 \quad 8E \quad 98 \quad 15 \quad 5B \quad D1 \quad E2 \quad 72 \quad 4E \quad 71 \\
D0 & 09 & 9A & A2 \quad 51 \quad 74 \quad F7 & 92 \quad D3 & 32 \quad 6C \quad 6F \quad 10 \quad 12 \quad 70 \quad 67 \\
1C & 62 & 13 & E3 \quad 93 \quad 0E \quad FD \quad DA & 26 & 45 & 17 \quad 92 \quad C6 & 20 \quad 81 \quad 22 \\
EE & 60 & D2 & 00 \quad 52 \quad 0D \quad 69 \quad 5D \quad FD \quad 9F & 5F \quad 0F \quad D5 \quad AB \quad A7 & 02 \\
\end{array} \\
\end{align*}
\]

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

\[
\begin{align*}
X &= \text{0x0185AE6271A81BB7F236A955F7CAAA26FB63849813C02B7D96C83A15AE6B6A864} \\
&= 67AB13B6D8CE8BCD7C2E5B97FF5F28FAC2C108F2A3CF3DB5515C9E6D7D210E8 \\
Y &= \text{0xEDB0220F92EF771A71CE64EC77986DB7C03D37B3E2AB3E83F32CE5E074A796EC0} \\
&= 8253C9E2102B87532661275C4B1D16D2789CDABC58ACFDF7318DE70AB64F09B8 \\
\end{align*}
\]

During the calculation of \( u_{\_1} \) on subject A, the parameter alpha, the point alpha*P, and \( u_{\_1} \) take the following values:

\[
\begin{align*}
\text{alpha} &= \text{0x332F930421D14CFE260042159F18E49FD5A54167E94108AD080BDE60B13DE799} \\
&= 9A34D611E63F3F870E5110247DF8E7C466E648ACF385E52CCB889ABF491EDFF0 \\
\text{alpha*P} &= \begin{array}{cccc}
X &= \text{0x561655966D52952E805574F4281F1ED3A2D498932B00CBA9DECBC42837F09835B} \\
&= FFBEF2D84DBB242FE7B57F92E1A6F2413E12D343E6437E13D76293469AD \\
Y &= \text{0xF6B18328B2715BD7F4178615273A36135BC0B62F7D8BF9080164AD36470AD} \\
&= 3660F51806C664C6691B3DEF30F793720F8E3FEAEDE631D6A54A4C72DCBF80E82 \\
\end{array} \\
\end{align*}
\]
u_1:
\[ X = 0x40645B4B9A908D74DEF98886A336F98BAE6ADA4C1AC9B7594A33D5E4A16486C5 \]
\[ 533C7F3C5DD48797AB5B4340BFC70CAF1011B69A01A715E5B9B5432D5151CBD7 \]
\[ Y = 0x267FBB18D0B79559D1875909F2A15F7B49ECD8ED166CF7F4FCD1F44891550483 \]
\[ 5E80D52BE8D34ADA5B5E5195C975981BCFE88F5048DC443A0983AA19192B8407 \]

When processing u_1, calculating the K_B key, and calculating u_2 on subject B, the parameters beta, src, K_B = HASH(src), beta*P, and u_2 take the following values:

beta=0x38481771E7D054F96212686B51338818B08A6C89DDBC656178F014D2C093432
A033EE10415F13A16D44C2AD61E62E05A7F7EC286BCEA3EA4D53F8634FA2

src:

60 A6 67 5E ED 42 1F C2 34 16 3F DE B4 4C 69 18
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
1D 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
E3 38 14 00 23 F3 92 E5 03 50 77 0E 3F B6 2C AC

K_B:
A0 83 84 A6 2F 4B E1 AE 48 98 FC A3 6D AA 3F AA
45 1B 3E C5 B5 9C E3 75 F8 9E 92 9F 4B 13 25 8C

beta*P:

X = 0xB7C581868708343BC1AFF61CB5CA79E38232025E0C1F12B3B651E62173CE687
3F3E6FFE78C2E45F4F524F66B0C263616ED08FD210AC4355CA3292B516D71C3
Y = 0x497F14205DBDC89BDDAF50528ED3B1429AD30777310186BE5E68070F016A44E0
C766DB08EBAC23FB2DFE66675AA4DF591EB18BA0D348DF7AA40973A2FD5CA55

u_2:

X = 0xB772FD97D6FDEC1DA0771BC059B3E5ADF9858311031EAE5AEC6A6EC8104B4105
C45A6C65689A8EE636C687DB62CC0AFC9A48CA66E381286CC73F374C1DD8F445
Y = 0xC64F69425FFEB2995130E85A08EDC3A686EC28EE6E8469F7F09BD3BCBDD843AC
573578DA6BA1CB3F5F069F205238535F06255C4B28586C9A1643537497B1018C

When processing u_2 and calculating the key on subject A, the K_A key takes the following values:

K_A:
A0 83 84 A6 2F 4B E1 AE 48 98 FC A3 6D AA 3F AA
45 1B 3E C5 B5 9C E3 75 F8 9E 92 9F 4B 13 25 8C

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

MAC_A:
12 63 F2 89 0E 90 EE 42 6B 9B A0 8A B9 EA 7F 1F
FF 26 E1 60 5C C6 5D E2 96 96 91 15 E5 31 76 87
The message \( MAC_B = \text{HMAC}(K_B, 0x02 || ID_B || \text{ind} || \text{salt} || u_1 || u_2) \) from subject B takes the following values:

\[
MAC_B:
\begin{align*}
6D & \text{ FD} \ 06 \ 04 \ 5D \ 97 \ A0 \ E4 \ 19 \ B0 \ 0E \ 00 \ 35 \ B9 \ D2 \\
E3 & \text{ AB} \ 09 \ 8B \ 7C \ A4 \ AD \ 52 \ 54 \ 60 \ FA \ B6 \ 21 \ B5 \ AA \ 57
\end{align*}
\]

Appendix B. Point Verification Script

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

curvesParams = [

    
    
    "OID":"id-GostR3410-2001-CryptoPro-A-ParamSet",
    "p":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD97,
    "a":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD94,
    "b":166,
    "m":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD96,
    "q":0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF6C611070995AD10045841B09B761B893,
    "x":1,
    "y":0x8D91E471E0989CDA27DF505A453F2B7635294F2DDF23E3B122ACC99C9E9F1E14,
    "n":32
},

    
    "OID":"id-GostR3410-2001-CryptoPro-B-ParamSet",
    "p":0x8000000000000000000000000000000000000000000000000000000000000C99,
    "a":0x8000000000000000000000000000000000000000000000000000000000000C96,
    "b":0x3E1AF419A269A5F866A7D3C25C3DF80AE979259373FF2B182F49DCE7E1BB8B,
    "m":0x800000000000000000000000000000000000000000000000000000000000015F700CFFFF1A624E5E497161BC8A198F,
    "q":0x800000000000000000000000000000000000000000000000000000000000015F700CFFFF1A624E5E497161BC8A198F,
    "x":1,
    "y":0xA8124359F96680B83D1C3EB2C070E5C545C9858D02ECFB744BF87D71777EFC,
    "n":32
},

    
    "OID":"id-GostR3410-2001-CryptoPro-C-ParamSet",
    "p":0xB9F605F5A858107AB1EC85E6B41C8AACC846E86789051D37998F7B90227D59B,
    "a":0xB9F605F5A858107AB1EC85E6B41C8AACC846E86789051D37998F7B90227D59B,
    "b":0xA858107AB1EC85E6B41C8AACC846E86789051D37998F7B90227D59B,
    "m":0x800000000000000000000000000000000000000000000000000000000000015F700CFFFF1A624E5E497161BC8A198F,
    "q":0x800000000000000000000000000000000000000000000000000000000000015F700CFFFF1A624E5E497161BC8A198F,
    "x":0,
    "y":0x41ECE55743711A8C3CBF3783CD08C0EE4D4DC440D4641A8F366E550DFDB3B67,
    "n":32
],
{
  "OID":"id-tc26-gost-3410-2012-512-paramSetA",
  "p":(0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296)+\
    (0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<80)+\
    0xFFFFFFFFFFFFFFFFFDC7L,
  "a":(0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296)+\n    (0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<80)+\n    0xFFFFFFFFFFFFFFFFFDC4L,
  "b":(0xE8C2505DEDFC86DDC1BD0B2B6667F1DA34B82574761CB0E879BD08L<<296)+\n    (0x1CFD0B6265EE3CB090F30D27614CB4574010DA90DD862F9D4EBEEL<<80)+\n    0x4761503190785A71C760L,
  "m":(0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296)+\n    (0xFFFFFFFFFF27E69532F48D9116FF22B84E0560609B4B38ABFAD2L<<80)+\n    0xB85DACADB1411F10B275L,
  "q":(0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296)+\n    (0xFFFFFFFFFF27E69532F48D9116FF22B84E0560609B4B38ABFAD2L<<80)+\n    0xB85DACADB1411F10B275L,
  "x":3,
  "y":(0x7503CFE87A836E3A61B8816E25450E6CE5E1C93ACF1ABC1778064L<<296)+\n    (0xFDCEBA921DF1626BE4FD03E93D75E6A50E3A41E98028E5FC235L<<80)+\n    0xF5B889A589CB5215F2A4L,
  "n":64
},
{
  "OID":"id-tc26-gost-3410-2012-512-paramSetB",
  "p":(0x80000000000000000000000000000000000000000000000000000L<<296)+\n    (0x00000000000000000000000000000000000000000000000000000L<<80)+\n    0x000000000000006FL,
  "a":(0x80000000000000000000000000000000000000000000000000000L<<296)+\n    (0x00000000000000000000000000000000000000000000000000000L<<80)+\n    0x000000000000006CL,
  "b":(0x687D1B459DC841457E3E06CF65E2517B97C7D614AF138BCBF85DCL<<296)+\n    (0x806C4B29BF3E96B5D2BD146D2178B276FAD1AB69C50F7B8E1FABA3L<<80)+\n    0x106EB6CCEB7C5140116L,
  "m":(0x80000000000000000000000000000000000000000000000000000L<<296)+\n    (0x000000000149A1EC142565A545ACFD877BD940CFA8899712101BL<<80)+\n    0xEA0EC346C474F25BDL,
  "q":(0x80000000000000000000000000000000000000000000000000000L<<296)+\n    (0x000000000149A1EC142565A545ACFD877BD940CFA8899712101BL<<80)+\n    0xEA0EC346C474F25BDL,
  "x":2,
  "y":(0x1A8F7EDA389B094C2C071E3647A8940F3123B697578C213B6ED99L<<296)+\n    (0xE6C8EC7335C5C8228DF1EFD4A39152CBBAA980398828041055F94CL<<80)+\n    0xEEEEC7E21340780FE41BDL,
  "n":64
},

Smyshlyaev, et al.            Informational                    [Page 34]
{
  "OID": "id-tc26-gost-3410-2012-256-paramSetA",
  "p": 0x1000000000000000000000000000000003F63377F21ED9B70456BD55B0DB8319C,
  "a": 0x295F9BAE7428ED9CEC267C359A941A22FCCD9108E17BF7B9A337A6F3AE9513,
  "m": 0x100000000000000000000000000000000F63377F21ED9B70456BD55B0DB8319C,
  "q": 0x4000000000000000000000000000000000000000000000000000000000000000FD8CDDFC87B6635C115AF556C360C67,
  "x": 0x91E38443A5E82C0D880923425712B2BB658B9196932E02C7B5282FE742DAAC,
  "y": 0x32879423AB1A0375895786C4BB46E9565FD0E9B33476740AF268ADB3222E5C,
  "n": 32
},
{
  "OID": "id-tc26-gost-3410-2012-512-paramSetC",
  "p": (0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296)+\(0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<80)+\0xFFFFFFFFFFFFFFFFFFFFDL<<32L,
  "a": (0xDC9203E514A7218754B5A529D2C722F187BC890EB86664DE41CL<<296)+\(0x68E143064546E861C0E29E2D92ADE71F4666CF50FF2AD97F951FDAL<<80)+\0x90F2A2B6546F39689DBC3L,
  "b": (0xB4C4EE28EC6C28AC12952CF37F16AC7EF66A9F694B57FFDA2EL<<296)+\(0x4F0DE5A5DE038C92CFF719D2C18DE028AB0B8BEE3B52B8CC7A5F59EL<<80)+\0x0A3CD58D331515257551EL,
  "m": (0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296)+\0xFFFFFFFFFFFFFFFFFFFFDL<<32L,
  "q": (0x3FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL<<296)+\0xFFFFFFFFFFFFC98CDBA46506AB004C33A9FF5147502CC8EDA9E7A769L<<80)+\0xA12694623CE5F0B4D59EL,
  "x": (0x2E21ED9B7DEBE241CE593EF5DE2295B79CA9B0EF021D385FF7L<<296)+\0x074C27272A78E602BFA7B9033DB9ED3610C6FB85487EAEL<<80)+\0x97A4C5B7928C1950148L,
  "y": (0xF5CE40D955B5EB899AABC4FF5911CB8577939804D6527378B8C108CL<<296)+\0x3D2090FF9BE18E2D33E3D3ED2EF32D85822423B6304F726AA854BL<<80)+\0xAEA07D0396E9A9ADD4CFL,
  "n": 64
}
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, 0x47107dd9b505a38, 0xad08b0e0c3282d1c,
        0xd8045870ef14980e, 0x6c022c38f90a4c07, 0x3601161cf205268d,
        0x83478b07b2468764, 0xa011d380818e8ff0, 0x5086e740ce47c920,
        0x2843fd2067adea10, 0x14aff010bdd87508, 0x0ad97808d06cb404,
        0x05e23c0468365a02, 0x8c71e02341b2d10, 0x46b60f011a83988e,
        0x90dab52a387ae76f, 0x486dd4151c3dfdb9, 0x24b86a840e90f0d2,
        0x125c354207487869, 0x092e94218d243cba, 0x8a174a9ec8121e5d,
        0x4585254f64090fa0, 0xacc9ca9328a8950, 0x9d4df05d5f661451,
        0xc0a0878a0a1330aa6, 0x6054350de70553, 0x0302a1e286fc58ca7,
        0x18150f14b9ec46dd, 0xc84890ad27623e0, 0x0642ca05693bf970,
        0x0321658cba93c138, 0x8e6275df09ce8aaa8, 0x439da074e745554,
        0xafc0503ca273aa42a, 0x9d60281e9d1d215, 0xece30140fc0802984,
        0x71180a89604094a2, 0xb60c05ca30204d21, 0x5b06b8e651a89e,
        0x456c34887a3805b9, 0xac361a443d1cced2, 0x561b0d22900e4669,
        0x2b838811480723ba, 0x9bce4f4862489df5d, 0xc3e9224312e8c1a0,
        0xeffaf1af09664ee50, 0xf97d86d9ba37278, 0xe4f4ea2054a80b329c,
        0x72d7c02a54b194e, 0x39b008152acb8227, 0x925804815e419d,
        0x492c024284fbaec0, 0xaaa16012142f35760, 0x5508e96e21f7aa530,
        0x4a8b474f9ef5dc18, 0x70a6a56e2440598e, 0x3853dc371220a247,
        0x1ca76e97091051ad, 0x0edd37c48a08a68, 0x07e095624504536c,
        0x8d70c431ac02a736, 0xc83862965601dd1b, 0x641c314b2b8ee083]
__Sbox = [
  0xFC, 0xEE, 0xDD, 0x11, 0xCF, 0x6E, 0x31, 0x16, 0xFB, 0xC4, 0xFA, 0xDA, 0x23, 0xC5, 0x04, 0x4D, 0xE9, 0x77, 0xF0, 0xDB, 0x93, 0x2E, 0x99, 0xBA, 0x17, 0x36, 0xF1, 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, 0x98, 0x7F, 0xD4, 0x9D, 0x9E, 0xB2, 0xB1, 0x32, 0x75, 0x19, 0x3D, 0xFF, 0x35, 0x8A, 0x7E, 0x6D, 0x54, 0xC6, 0x80, 0xC3, 0x90, 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, 0x4A, 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, 0x1D, 0xF7, 0x30, 0x37, 0xB, 0xE4, 0x88, 0xD9, 0xE7, 0x89, 0xB1, 0x83, 0x49, 0x4C, 0x3F, 0xF8, 0xFE, 0xD8, 0x53, 0xAA, 0x90, 0xCA, 0xD8, 0x85, 0x61, 0x20, 0x71, 0x67, 0xA4, 0x2D, 0x2B, 0x09, 0xB2, 0xCB, 0x9B, 0x25, 0xD0, 0xBE, 0xE5, 0x6C, 0x52, 0x59, 0xA6, 0x74, 0xD2, 0xE6, 0x4F, 0xB4, 0xC0, 0xD1, 0x66, 0xAF, 0x39, 0x4B, 0x63, 0xB6
]

__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
]
__C = [
  [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, 0x08, 0xa4, 0x60, 0x09, 0x15, 0x05,
   0x76, 0x74, 0x36, 0x4c, 0x74, 0x4d, 0x23, 0xdd,
   0x80, 0x65, 0x59, 0xf2, 0xa6, 0x45, 0x07],
  [0x6f, 0xa3, 0xb5, 0x8a, 0xa9, 0x9d, 0x2f, 0x1a,
   0x4f, 0xe3, 0x9d, 0x46, 0x0f, 0x70, 0x0f, 0x70,
   0x0b5, 0xd3, 0x15, 0x05, 0x76, 0x36, 0xcc, 0x74,
   0x4b, 0x7c, 0xe0, 0x91, 0x92, 0x67, 0x69, 0x01,
   0xa2, 0x42, 0x08, 0xa4, 0x60, 0x09, 0x15, 0x05,
   0x76, 0x74, 0x36, 0x4c, 0x74, 0x4d, 0x23, 0xdd,
   0x80, 0x65, 0x59, 0xf2, 0xa6, 0x45, 0x07],
  [0xf5, 0x74, 0xdc, 0xac, 0x2b, 0x0c, 0xe, 0x2f,
   0x7c, 0x9a, 0x39, 0xfc, 0x28, 0x6a, 0x3d, 0x84,
   0x35, 0x06, 0xf1, 0x5e, 0x5f, 0x52, 0x9c, 0x1f,
   0x8b, 0x0f, 0x70, 0x0b1, 0x0b2, 0x0f, 0x70,
   0x0b1, 0x0b2, 0x0f, 0x70, 0x0b1, 0x0b2, 0x0f],
  [0xf9, 0x48, 0xe1, 0xa0, 0x5d, 0x71, 0xe4, 0x33,
   0x5c, 0x33, 0x7d, 0x0d, 0x72, 0x1c, 0xad, 0x68,
   0x5e, 0x35, 0x3f, 0x9a, 0x4d, 0x82, 0xed, 0x03,
   0xda, 0x4b, 0xe9, 0x37, 0x93, 0x52, 0xe1, 0x3F,
   0x34, 0x35, 0x0a, 0x1a, 0x93, 0xe8, 0x37, 0x9e,
   0xb2, 0x0c, 0xb6, 0x9e, 0x96, 0xf5, 0x0a, 0x0a,
   0x0b2, 0x0f, 0x70, 0x0b1, 0x0b2, 0x0f, 0x70,
   0x0b1, 0x0b2, 0x0f, 0x70, 0x0b1, 0x0b2, 0x0f],
]
[0x4b, 0xea, 0xac, 0xad, 0x47, 0x47, 0x99, 0x9a, 0x3f, 0x41, 0x0c, 0x6c, 0xa9, 0x23, 0x63, 0x7f, 0x15, 0x1c, 0x1f, 0x16, 0x86, 0x10, 0x4a, 0x35, 0x9e, 0x35, 0xd7, 0x80, 0x0f, 0xff, 0xb0, 0xc0, 0x2d, 0x66, 0xc4, 0xf9, 0x51, 0x42, 0xa4, 0x6c, 0x47, 0x25, 0x3a, 0xf5, 0xa3, 0xdf, 0xff, 0x00, 0xb7, 0x23, 0x27, 0x1a, 0x16, 0x7a, 0x56, 0xa2, 0x7e, 0xa9, 0xea, 0x63, 0xf5, 0x4a, 0x35, 0x9e, 0x35, 0xd7, 0x80, 0x0f, 0xff, 0xbd, 0xbf, 0xcd, 0x17, 0x47, 0x25, 0x3a, 0xf5, 0xa3, 0xdf, 0xff, 0x00, 0xb7, 0x23, 0x27, 0x1a, 0x16, 0x7a, 0x56, 0xa2, 0x7e, 0xa9, 0xea, 0x63, 0xf5, 0x60, 0x17, 0x58, 0xfd, 0x7c, 0x6c, 0xfe, 0x57, 0x5f, 0x9a, 0xea, 0x63, 0xf5, 0x60, 0x17, 0x58, 0xfd, 0x7c, 0x6c, 0xfe, 0x57, 0x4f, 0xae, 0xae, 0x1d, 0x3a, 0x3a, 0x9d, 0x6f, 0xa4, 0xc3, 0x3b, 0x7a, 0x30, 0x39, 0xc0, 0x2d, 0x66, 0xc4, 0xf9, 0x51, 0x42, 0xa4, 0x6c, 0x18, 0x7f, 0x9a, 0xb4, 0x9a, 0xf0, 0x8e, 0xc6, 0xcf, 0xfa, 0xa6, 0xb7, 0x1c, 0x9a, 0xb7, 0xb4, 0x0a, 0xf2, 0x1f, 0x66, 0xc2, 0xbe, 0xc6, 0xb6, 0xbf, 0x71, 0xc5, 0x72, 0x36, 0x90, 0x4f, 0x35, 0xfa, 0x68, 0x40, 0x7a, 0x46, 0x64, 0x7d, 0x6e, 0xf4, 0xc7, 0x0e, 0x16, 0xea, 0x0a, 0xc5, 0xec, 0x51, 0x9c, 0x86, 0xe6, 0xbf, 0x24, 0x09, 0x54, 0x39, 0x9c, 0x86, 0xc7, 0xe6, 0xbf, 0x87, 0xc9, 0x3d, 0x47, 0x3e, 0x33, 0x19, 0x7a, 0x93, 0xc9, 0x09, 0x92, 0xc5, 0x2d, 0x32, 0x2c, 0x2d, 0x47, 0x69, 0x83, 0x28, 0x4a, 0x05, 0x04, 0x35, 0x17, 0x45, 0x4c, 0xa2, 0x3c, 0x4a, 0x4f, 0x88, 0x86, 0x56, 0x4d, 0x33, 0x14, 0xd4, 0x93, 0x9b, 0x1f, 0x5b, 0x42, 0xa7, 0x03, 0xe7, 0x0a, 0x02, 0x6e, 0x41, 0x41, 0x4e, 0xb7, 0xf8, 0x71, 0x9c, 0x36, 0xde, 0x1e, 0x89, 0xb4, 0x44, 0x3b, 0x4d, 0x3b, 0x49, 0x4f, 0x89, 0x2b, 0xc0, 0x92, 0x9b, 0x06, 0x90, 0x60, 0x71, 0x8d, 0x2b, 0x6d, 0xa5, 0xc4, 0x2f, 0x36, 0xe6, 0x35, 0x59, 0x51, 0xa8, 0x9d, 0xa4, 0x7f, 0x0d, 0x4d, 0x8b, 0x02, 0xe7, 0x1e]
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)
N = 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 = powF(R, (s + 1) / 2, p)

        bj = 1
        for k in range(0, alpha - 1):  # alpha >= 2 because p % 4 = 1
            d = 2 ** (alpha - k - 2)
            x = powF(mulF(powF(mulF(bj, r, p), 2, p), ainvF, p), d, p)
            if x != 1:
                bj = mulF(bj, powF(b, (2 ** k), p), p)
        res = mulF(bj, r, p)
        return [res, negF(res, p)]
def generateQs( p, pByteSize, a, b, m, q, orderDivisors, Px, Py, N):
    assert pByteSize in ( 256 / 8, 512 / 8 )
    PxBytes = num2le( Px, pByteSize )
    PyBytes = num2le( Py, pByteSize )
    Qs = []
    S = []
    Hash_src = []
    Hash_res = []
    co_factor = m / q

    seed = 0
    while len( Qs ) != N:
        hashSrc = PxBytes + PyBytes + num2le( seed, 4 )
        if pByteSize == ( 256 / 8 ):
            QxBytes = H256( hashSrc )
        else:
            QxBytes = H512( hashSrc )

        Qx = le2num( QxBytes ) % p
        R = addF( addF( powF(Qx, 3, p ), mulF(Qx, a, p), p), b, p )
        if ( R == 0 ) or ( not isQuadraticResidue( R, p ) ) :
            seed += 1
            continue

        Qy_sqrt = 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

        Smyshlyaev, et al. Informational [Page 50]
Acknowledgments

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

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