Internet Engineering Task Force (IETF)
Request for Comments: 6507
Category: Informational
ISSN: 2070-1721

M. Groves
CESG

February 2012

Elliptic Curve-Based Certificateless Signatures for Identity-Based Encryption (ECCSI)

Abstract

Many signature schemes currently in use rely on certificates for authentication of identity. In Identity-based cryptography, this adds unnecessary overhead and administration. The Elliptic Curve-based Certificateless Signatures for Identity-based Encryption (ECCSI) signature scheme described in this document is certificateless. This scheme has the additional advantages of low bandwidth and low computational requirements.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Digital signatures provide authentication services across a wide range of applications. A chain of trust for such signatures is usually provided by certificates. However, in low-bandwidth or other resource-constrained environments, the use of certificates might be undesirable. This document describes an efficient scheme, ECCSI, for elliptic curve-based certificateless signatures, primarily intended for use with Identity-Based Encryption (IBE) schemes such as described in [RFC6508]. As certificates are not needed, the need to transmit or store them to authenticate each communication is obviated. The algorithm has been developed by drawing on ideas set out by Arazi [BA] and is originally based upon the Elliptic Curve Digital Signature Algorithm [ECDSA], one of the most commonly used signature algorithms.

The algorithm is for use in the following context:

* where there are two parties, a Signer and a Verifier;

* where short unambiguous Identifier strings are naturally associated to each of these parties;

- where a message is to be signed and then verified (e.g., for authenticating the initiating party during an Identity-based key establishment);
- where a common Key Management Service (KMS) provides a root of trust for both parties.

The scheme does not rely on any web of trust between users.

Authentication is provided in a single simplex transmission without per-session reference to any third party. Thus, the scheme is particularly suitable in situations where the receiving party need not be active (or even enrolled) when the message to be authenticated is sent, or where the number of transmissions is to be minimized for efficiency.

Instead of having a certificate, the Signer has an Identifier, to which his Secret Signing Key (SSK) (see Section 2) will have been cryptographically bound by means of a Public Validation Token (PVT) (see Section 2) by the KMS. Unlike a traditional public key, this PVT requires no further explicit certification.

The verification primitive within this scheme can be implemented using projective representation of elliptic curve points, without arithmetic field divisions, and without explicitly using the size of the underlying cryptographic group.

1.1. Requirements Terminology

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].

2. Architecture

A KMS provisions key material for a set of communicating devices (a "user community"). Each device within the user community MUST have an Identifier (ID), which can be formed by its peers. These Identifiers MUST be unique to devices (or users), and MAY change over time. As such, all applications of this signature scheme MUST define an unambiguous format for Identifiers. We consider the situation where one device (the Signer) wishes to sign a message that it is sending to another (the Verifier). Only the Signer’s ID is used in the signature scheme.

In advance, the KMS chooses its KMS Secret Authentication Key (KSAK), which is the root of trust for all other key material in the scheme. From this, the KMS derives the KMS Public Authentication Key (KPAK), which all devices will require in order to verify signatures. This will be the root of trust for verification.

Before verification of any signatures, members of the user community are supplied with the KPAK. The supply of the KPAK MUST be authenticated by the KMS, and this authentication MUST be verified by each member of the user community. Confidentiality protection MAY also be applied.

In the description of the algorithms in this document, it is assumed that there is one KMS, one user community, and hence one KPAK. Applications MAY support multiple KPAKs, and some KPAKs could in fact be "private" to certain communities in certain circumstances. The method for determining which KPAK to use (when more than one is available) is out of scope.

The KMS generates and provisions key material for each device. It MUST supply an SSK along with a PVT to all devices that are to send signed messages. The mechanism by which these SSKs are provided MUST be secure, as the security of the authentication provided by ECCSI signatures is no stronger than the security of this supply channel.

Before using the supplied key material (SSK, KPAK) to form signatures, the Sender MUST verify the key material (SSK) against the root of trust (KPAK) and against its own ID and its PVT, using the algorithm defined in Section 5.1.2.

During the signing process, once the Signer has formed its message, it signs the message using its SSK. It transmits the Signature (including the PVT), and MAY also transmit the message (in cases where the message is not known to the Verifier). The Verifier MUST then use the message, Signature, and Sender ID in verification against the KPAK.

This document specifies

- an algorithm for creating a KPAK from a KSAK, for a given elliptic curve;

- a format for transporting a KPAK;

- an algorithm for creating an SSK and a PVT from a Signer’s ID, using the KSAK;

- an algorithm for verifying an SSK and a PVT against a Signer’s ID and KPAK;
- an algorithm for creating a Signature from a message, using a Signer’s ID with a matching SSK and PVT;
- a format for transporting a Signature;
- an algorithm for verifying a Signature for a message, using a Signer’s ID with the matching KPAK.

This document does not specify (but comments on)

- how to choose a valid and secure elliptic curve;
- which hash function to use;
- how to format a Signer’s ID;
- how to format a message for signing;
- how to manage and install a KPAK;
- how to transport or install an SSK.

As used in [RFC6509], the elliptic curve and hash function are specified in Section 2.1.1 of [RFC6509], the format of Identifiers is specified in Section 3.2 of [RFC6509], and messages for signing are formatted as specified in [RFC3830].

3. Notation

3.1. Arithmetic

ECCSI relies on elliptic curve arithmetic. If P and Q are two elliptic curve points, their addition is denoted P + Q. Moreover, the addition of P with itself k times is denoted [k]P.

F_p denotes the finite field of p elements, where p is prime. All elliptic curve points will be defined over F_p.

The curve is defined by the equation y^2 = x^3 - 3*x + B modulo p, where B is an element of F_p. Elliptic curve points, other than the group identity (0), are represented in the format P = (Px,Py), where Px and Py are the affine coordinates in F_p satisfying the above equation. In particular, a point P = (Px,Py) is said to lie on an elliptic curve if Py^2 - Px^3 + 3*Px - B = 0 modulo p. The identity point 0 will require no representation.

3.2. Representations

This section provides canonical representations of values that MUST be used to ensure interoperability of implementations. The following representations MUST be used for input into hash functions and for transmission. In this document, concatenation of octet strings s and t is denoted s || t. The logarithm base 2 of a real number a is denoted lg(a).

Integers	Integers MUST be represented as an octet string, with bit length a multiple of 8. To achieve this, the integer is represented most significant bit first, and padded with zero bits on the left until an octet string of the necessary length is obtained. This is the octet string representation described in Section 6 of [RFC6090]. There will be no need to represent negative integers. When transmitted or hashed, such octet strings MUST have length N = Ceiling(lg(p)/8).
F_p elements	Elements of F_p MUST be represented as integers in the range 0 to p-1 using the octet string representation defined above. For use in ECCSI, such octet strings MUST have length N = Ceiling(lg(p)/8).
Points on E	Elliptic curve points MUST be represented in uncompressed form ("affine coordinates") as defined in Section 2.2 of [RFC5480]. For an elliptic curve point (x,y) with x and y in F_p, this representation is given by 0x04 || x’ || y’, where x’ is the N-octet string representing x and y’ is the N-octet string representing y.

3.3. Format of Material

This section describes the subfields of the different objects used within the protocol.

Signature = r || s || PVT	where r and s are octet strings of length N = Ceiling(lg(p)/8) representing integers, and PVT is an octet string of length 2N+1 representing an elliptic curve point, yielding a total signature length of 4N+1 octets. (Note that r and

s represent integers rather than elements of F_p, and therefore it is possible that either or both of them could equal or exceed p.)

4. Parameters

4.1. Static Parameters

The following static parameters are fixed for each implementation. They are not intended to change frequently, and MUST be specified for each user community. Note that these parameters MAY be shared across multiple KMSs.

n	A security parameter; the size in bits of the prime p over which elliptic curve cryptography is to be performed.
N = Ceiling(n/8)	The number of octets used to represent fields r and s in a Signature. Also the number of octets output by the hash function (see below).
p	A prime number of size n bits. The finite field with p elements is denoted F_p.
E	An elliptic curve defined over F_p, having a subgroup of prime order q.
B	An element of F_p, where E is defined by the formula y^2 = x^3 - 3*x + B modulo p.
G	A point on the elliptic curve E that generates the subgroup of order q.
q	The prime q is defined to be the order of G in E over F_p.
Hash	A cryptographic hash function mapping arbitrary strings to strings of N octets. If a, b, c, ... are strings, then hash(a || b || c || ...) denotes the result obtained by hashing the concatenation of these strings.
Identifiers	The method for deriving user Identifiers. The format of Identifiers MUST be specified by each implementation. It MUST be possible for each device to derive the Identifier for every device with which it needs to communicate. In

this document, ID will denote the correctly formatted Identifier string of the Signer. ECCSI makes use of the Signer Identifier only, though an implementation MAY make use of other Identifiers when constructing the message to be signed. Identifier formats MAY include a timestamp to allow for automatic expiration of key material.

It is RECOMMENDED that p, E, and G are chosen to be standardized values. In particular, it is RECOMMENDED that the curves and base points defined in [FIPS186-3] be used.

4.2. Community Parameters

The following community parameter MUST be supplied to devices each time the root of trust is changed.

KPAK	The KMS Public Authentication Key (KPAK) is the root of trust for authentication. It is derived from the KSAK in the KMS. This value MUST be provisioned in a trusted fashion, such that each device that receives it has assurance that it is the genuine KPAK belonging to its KMS. Before use, each device MUST check that the supplied KPAK lies on the elliptic curve E.

The KMS MUST fix the KPAK to be KPAK = [KSAK]G, where the KSAK MUST be chosen to be a random secret non-zero integer modulo q. The value of the KSAK MUST be kept secret to the KMS.

5. Algorithms

5.1. User Key Material

To create signatures, each Signer requires a Secret Signing Key (SSK) and a Public Validation Token (PVT). The SSK is an integer, and the PVT is an elliptic curve point. The SSK MUST be kept secret (to the Signer and KMS), but the PVT need not be kept secret. A different (SSK,PVT) pair will be needed for each Signer ID.

5.1.1. Algorithm for Constructing (SSK,PVT) Pair

The KMS constructs a (SSK,PVT) pair from the Signer’s ID, the KMS secret (KSAK), and the root of trust (KPAK). To do this, the KMS MUST perform the following procedure:

1) Choose v, a random (ephemeral) non-zero element of F_q;

2) Compute PVT = [v]G (this MUST be represented canonically -- see Section 3.2);

3) Compute a hash value HS = hash(G || KPAK || ID || PVT), an N-octet integer;

4) Compute SSK = (KSAK + HS * v) modulo q;

5) If either the SSK or HS is zero modulo q, the KMS MUST erase the SSK and abort or restart the procedure with a fresh value of v;

6) Output the (SSK,PVT) pair. The KMS MUST then erase the value v.

The method for transporting the SSK to the legitimate Signer device is out of scope for this document, but the SSK MUST be provisioned by the KMS using a method that protects its confidentiality.

If necessary, the KMS MAY create multiple (SSK,PVT) pairs for the same Identifier.

5.1.2. Algorithm for Validating a Received SSK

Every SSK MUST be validated before being installed as a signing key. The Signer uses its ID and the KPAK to validate a received (SSK,PVT) pair. To do this validation, the Signer MUST perform the following procedure, passing all checks:

1) Validate that the PVT lies on the elliptic curve E;

2) Compute HS = hash(G || KPAK || ID || PVT), an N-octet integer. The integer HS SHOULD be stored with the SSK for later use;

3) Validate that KPAK = [SSK]G - [HS]PVT.

5.2. Signatures

5.2.1. Algorithm for Signing

To sign a message (M), the Signer requires

* the KMS Public Authentication Key, KPAK;

* the Signer’s own Identifier, ID;

- its Secret Signing Key, SSK;
- its Public Validation Token, PVT = (PVTx,PVTy).

These values, with the exception of ID, MUST have been provided by the KMS. The value of ID is derived by the Signer using the community-defined method for formatting Identifiers.

The following procedure MUST be used by the Signer to compute the signature:

1) Choose a random (ephemeral) non-zero value j in F_q;

2) Compute J = [j]G (this MUST be represented canonically). Viewing J in affine coordinates J = (Jx,Jy), assign to r the N-octet integer representing Jx;

3) Recall (or recompute) HS, and use it to compute a hash value HE = hash(HS || r || M);

4) Verify that HE + r * SSK is non-zero modulo q; if this check fails, the Signer MUST abort or restart this procedure with a fresh value of j;

5) Compute s’ = (((HE + r * SSK)^-1) * j) modulo q; the Signer MUST then erase the value j;

6) If s’ is too big to fit within an N-octet integer, then set the N-octet integer s = q - s’; otherwise, set the N-octet integer s = s’ (note that since p is less than 2^n, by Hasse’s theorem on elliptic curves, q < 2^n + 2^(n/2 + 1) + 1. Therefore, if s’ > 2^n, we have q - s’ < 2(n/2 + 1) + 1. Thus, s is guaranteed to fit within an N-octet integer);

7) Output the signature as Signature = (r || s || PVT).

Note that step 6) is necessary because it is possible for q (and hence for elements of F_q) to be too big to fit within N octets. The Signer MAY instead elect to set s to be the least integer of s’ and q - s’, represented in N octets.

5.2.2. Algorithm for Verifying

The algorithm provided assumes that the Verifier computes points on elliptic curves using affine coordinates. However, the Verifier MAY perform elliptic curve operations using any appropriate representation of points that achieves the equivalent operations.

To verify a Signature (r || s || PVT) against a Signer’s Identifier (ID), a message (M), and a pre-installed root of trust (KPAK), the Verifier MUST perform a procedure equivalent to the following:

1) The Verifier MUST check that the PVT lies on the elliptic curve E;

2) Compute HS = hash(G || KPAK || ID || PVT);

3) Compute HE = hash(HS || r || M);

4) Y = [HS]PVT + KPAK;

5) Compute J = [s]([HE]G + [r]Y);

6) Viewing J in affine coordinates (Jx,Jy), the Verifier MUST check that Jx = r modulo p, and that Jx modulo p is non-zero, before accepting the Signature as valid.

It is anticipated that the ID, message (M), and KPAK will be implicitly understood due to context, but any of these values MAY also be included in signaling.

Note that the parameter q is not needed during verification.

6. Security Considerations

The ECCSI cryptographic algorithm is based upon [ECDSA]. In fact, step 5) in the verification algorithm above is the same as the verification stage in ECDSA. The only difference between ECDSA and ECCSI is that in ECCSI the ’public key’, Y, is derived from the Signer ID by the Verifier (whereas in ECDSA the public key is fixed). It is therefore assumed that the security of ECCSI depends entirely on the secrecy of the secret keys. In addition, to recover secret keys, one will need to perform computationally intensive cryptanalytic attacks.

The KSAK provides the security for each device provisioned by the KMS. It MUST NOT be revealed to any entity other than the KMS that holds it. Each user’s SSK authenticates the user as being associated with the ID to which the SSK is assigned by the KMS. This key MUST NOT be revealed to any entity other than the KMS and the authorized user.

The order of the base point G used in ECCSI MUST be a large prime q. If k bits of symmetric security are needed, Ceiling(lg(q)) MUST be at least 2*k.

It is RECOMMENDED that the curves and base points defined in [FIPS186-3] be used, since these curves are suitable for cryptographic use. However, if other curves are used, the security of the curves MUST be assessed.

In order to ensure that the SSK is only received by an authorized device, it MUST be provided through a secure channel. The strength of the authentication offered by this signature scheme is no greater than the security provided by this delivery channel.

Identifiers MUST be defined unambiguously by each application of ECCSI. Note that it is not necessary to use a hash function to compose an Identifier string. In this way, any weaknesses that might otherwise be caused by collisions in hash functions can be avoided without reliance on the structure of the Identifier format. Applications of ECCSI MAY include a time/date component in their Identifier format to ensure that Identifiers (and hence SSKs) are only valid for a fixed period of time.

The use of the ephemeral value r in the hash HE significantly reduces the scope for offline attacks, improving the overall security, as compared to [ECDSA]. Furthermore, if Identifiers are specified to contain date-stamps, then all Identifiers, SSKs, signatures, and hash values will periodically become deprecated automatically, reducing the need for revocation and other additional management methods.

The randomness of values stipulated to be selected at random, as described in this document, is essential to the security provided by ECCSI. If the value of the KSAK can be predicted, then any signatures can be forged. Similarly, if the value of v used by the KMS to create a user’s SSK can be predicted, then the value of the KSAK could be recovered, which would allow signatures to be forged. If the value of j used by a user is predictable, then the value of his SSK could be recovered. This would allow that user’s signatures to be forged. Guidance on the generation of random values for security can be found in [RFC4086].

Note that in most instances, the value s in the Signature can be replaced by q - s. Thus, the malleability of ECCSI signatures is similar to that in [ECDSA]; malleability is available but also very limited.

7. References

7.1. Normative References

[ECDSA] X9.62-2005, "Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", November 2005.

[FIPS186-3] Federal Information Processing Standards Publication (FIPS PUB) 186-3, "Digital Signature Standard (DSS)", June 2009.

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

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

7.2. Informative References

[BA] Arazi, Benjamin, "Certification of DL/EC Keys", paper submitted to P1363 meeting, August 1998, <http://grouper.ieee.org/groups/1363/StudyGroup/contributions/arazi.doc>.

[FIPS180-3] Federal Information Processing Standards Publication (FIPS PUB) 180-3, "Secure Hash Standard (SHS)", October 2008.

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC6508] Groves, M., "Sakai-Kasahara Key Encryption (SAKKE)", RFC 6508, February 2012.

[RFC6509] Groves, M., "MIKEY-SAKKE: Sakai-Kasahara Key Encryption in Multimedia Internet KEYing (MIKEY)", RFC 6509, February 2012.

Appendix A. Test Data

This appendix provides test data built from the NIST P-256 curve and base point. SHA-256 (as defined in [FIPS180-3]) is used as the hash function. The keys and ephemerals -- KSAK, v, and j -- are arbitrary and for illustration only.

```
// ----------------------------------------------------------
// Global parameters

n       := 256;

N       := 32;

p       := 0x   FFFFFFFF 00000001 00000000 00000000
                00000000 FFFFFFFF FFFFFFFF FFFFFFFF;

Hash    := SHA-256;

// ----------------------------------------------------------
// Community parameters

B       := 0x   5AC635D8 AA3A93E7 B3EBBD55 769886BC
                651D06B0 CC53B0F6 3BCE3C3E 27D2604B;

q       := 0x   FFFFFFFF 00000000 FFFFFFFF FFFFFFFF
                BCE6FAAD A7179E84 F3B9CAC2 FC632551;

G       := 0x   04
                6B17D1F2 E12C4247 F8BCE6E5 63A440F2
                77037D81 2DEB33A0 F4A13945 D898C296
                4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16
                2BCE3357 6B315ECE CBB64068 37BF51F5;

KSAK    := 0x   12345;

KPAK    := 0x   04
                50D4670B DE75244F 28D2838A 0D25558A
                7A72686D 4522D4C8 273FB644 2AEBFA93
                DBDD3755 1AFD263B 5DFD617F 3960C65A
                8C298850 FF99F203 66DCE7D4 367217F4;

// ----------------------------------------------------------
// Signer ID

ID      := "2011-02\0tel:+447700900123\0",
         = 0x   3230 31312D30 32007465 6C3A2B34
                34373730 30393030 31323300;
```

```
      // --------------------------------------------------------
      // Creating SSK and PVT

      v       := 0x   23456;

      PVT     := 0x   04
                      758A1427 79BE89E8 29E71984 CB40EF75
                      8CC4AD77 5FC5B9A3 E1C8ED52 F6FA36D9
                      A79D2476 92F4EDA3 A6BDAB77 D6AA6474
                      A464AE49 34663C52 65BA7018 BA091F79;

      HS      := hash( 0x 04
                          6B17D1F2 E12C4247 F8BCE6E5 63A440F2
                          77037D81 2DEB33A0 F4A13945 D898C296
                          4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16
                          2BCE3357 6B315ECE CBB64068 37BF51F5
                          04
                          50D4670B DE75244F 28D2838A 0D25558A
                          7A72686D 4522D4C8 273FB644 2AEBFA93
                          DBDD3755 1AFD263B 5DFD617F 3960C65A
                          8C298850 FF99F203 66DCE7D4 367217F4
                          32303131 2D303200 74656C3A 2B343437

                          37303039 30303132 3300
                          04
                          758A1427 79BE89E8 29E71984 CB40EF75
                          8CC4AD77 5FC5B9A3 E1C8ED52 F6FA36D9
                          A79D2476 92F4EDA3 A6BDAB77 D6AA6474
                          A464AE49 34663C52 65BA7018 BA091F79 ),

               = 0x       490F3FEB BC1C902F 6289723D 7F8CBF79
                          DB889308 49D19F38 F0295B5C 276C14D1;

      SSK     := 0x       23F374AE 1F4033F3 E9DBDDAA EF20F4CF
                          0B86BBD5 A138A5AE 9E7E006B 34489A0D;

      // --------------------------------------------------------
      // Creating a Signature

      M       := "message\0",
               = 0x   6D657373 61676500;

      j       := 0x   34567;
```

```
J       := 0x   04
                269D4C8F DEB66A74 E4EF8C0D 5DCC597D
                DFE6029C 2AFFC493 6008CD2C C1045D81
                6DDA6A13 10F4B067 BD5DABDA D741B7CE
                F36457E1 96B1BFA9 7FD5F8FB B3926ADB;

r       := 0x   269D4C8F DEB66A74 E4EF8C0D 5DCC597D
                DFE6029C 2AFFC493 6008CD2C C1045D81;

HE      := hash( 0x
                490F3FEB BC1C902F 6289723D 7F8CBF79
                DB889308 49D19F38 F0295B5C 276C14D1
                269D4C8F DEB66A74 E4EF8C0D 5DCC597D
                DFE6029C 2AFFC493 6008CD2C C1045D81
                6D657373 61676500 ),

         = 0x   111F90EA E8271C96 DF9B3D67 26768D9E
                E9B18145 D7EC152C FA9C23D1 C4F02285;

s’      := 0x   E09B528D 0EF8D6DF 1AA3ECBF 80110CFC
                EC9FC682 52CEBB67 9F413484 6940CCFD;

s       := 0x   E09B528D 0EF8D6DF 1AA3ECBF 80110CFC
                EC9FC682 52CEBB67 9F413484 6940CCFD;

Sig     := 0x   269D4C8F DEB66A74 E4EF8C0D 5DCC597D
                DFE6029C 2AFFC493 6008CD2C C1045D81
                E09B528D 0EF8D6DF 1AA3ECBF 80110CFC
                EC9FC682 52CEBB67 9F413484 6940CCFD
                04

                758A1427 79BE89E8 29E71984 CB40EF75
                8CC4AD77 5FC5B9A3 E1C8ED52 F6FA36D9
                A79D2476 92F4EDA3 A6BDAB77 D6AA6474
                A464AE49 34663C52 65BA7018 BA091F79;

// -------------------------------------------------------
// Verifying a Signature

Y       := 0x   04
                833898D9 39C0013B B0502728 6F95CCE0
                37C11BD2 5799423C 76E48362 A4959978
                95D0473A 1CD6186E E9F0C104 B472499E
                1A24D6CE 3D85173F 02EBBD94 5C25F604;
```

```
      J       := 0x   04
                      269D4C8F DEB66A74 E4EF8C0D 5DCC597D
                      DFE6029C 2AFFC493 6008CD2C C1045D81
                      6DDA6A13 10F4B067 BD5DABDA D741B7CE
                      F36457E1 96B1BFA9 7FD5F8FB B3926ADB;

      Jx      := 0x   269D4C8F DEB66A74 E4EF8C0D 5DCC597D
                      DFE6029C 2AFFC493 6008CD2C C1045D81;

      Jx = r  modulo p

      // -------------------------------------------------------
```

Author's Address

Michael Groves
CESG
Hubble Road
Cheltenham
GL51 8HJ
UK

EMail: Michael.Groves@cesg.gsi.gov.uk