INTERNET-DRAFT
Intended Status: Informational
Expires: March 15, 2014

L. Hitt
21CT, Inc.

September 11, 2013

ZSS Short Signature Scheme for BN Curves
draft-irtf-cfrg-zssbn-01

Abstract

This document describes the ZSS Short Signature Scheme for implementation from bilinear pairings on Barreto-Naerhig (BN) ordinary elliptic curves. The ZSS Short Signature Scheme uses general cryptographic hash functions such as SHA-1 or SHA-2 and is efficient in terms of pairing operations.

Status of this Memo

Copyright and License Notice

Table of Contents

1 Introduction 3
1.1 Bilinear Pairings 4
1.2 Discrete Logarithm Problem and Diffie-Hellman Problems . . . 4
1.3 Terminology 5
2 Architecture 5
3 Notation, Definitions and Parameters 6
3.1 Notation 6
3.2 Definitions 7
3.3 Representations 8
3.4 Arithmetic 8
4 The ZSS Cryptosystem 9
4.1 Parameter Generation 9
4.2 Key Generation 9
4.3 Signature Generation 9
4.4 Signature Verification 10
5 Security Considerations 10
6 IANA Considerations 11
7 References 12
7.1 Normative References 12
7.2 Informative References 12
Appendix A. BN Elliptic Curves, Twists, Pairings and Supporting
Algorithms 14
A.1. BN Elliptic Curves 14
A.2. Sextic Twists 14
A.3. The Ate Pairing 14
A.4. Hashing to an Integer Range 16
Appendix B. Example Data 17
B.1 Example 1 17
B.2 Example 2 19
Author's Address 21

1 Introduction

This document describes the ZSS Short Signature Scheme (designed by Zhang, Safavi-Naini, and Susilo) for implementation from bilinear pairings [ZSS]. It does not require any special hash function such as MapToPoint [B-F], which is still probabilistic and generally inefficient, but rather can use cryptographic hash functions such as SHA-1 or SHA-2.

This document is restricted to implementation of ZSS on a particular family of Barreto-Naerhig (BN) elliptic curves, though the scheme is valid on other elliptic curve groups. BN curves are a family of non-supersingular (i.e., ordinary) curves that are attractive for pairing-based cryptography for a variety of reasons. These curves are plentiful and easily found and they support a sextic twist, which allows pairing arguments to be defined over relatively small finite fields. Computation of the pairing is the most time consuming procedure in pairing-based cryptography, and BN curves are amenable to twofold or threefold pairing compression and attain high efficiency for all pairing computation algorithms known (e.g., Tate, ate, eil, R-ate, Xate). These curves are also suitable for software and hardware implementations on a wide range of platforms.

The specific subclass of BN curves that we choose for this document is discussed in [Pereira], and offers many additional efficiency advantages. The subclass automatically yields the right sextic twist (thus entirely avoiding curve arithmetic for that purpose) and provides simple and obvious generators for the curve and its twist (removing the need for extra processing and storage). It allows for pairing efficiency, uniform finite field arithmetic, choice of suitable field sizes, and enables virtually all optimizations currently proposed in the literature for involved algebraic structures. These advantages are important since short signatures are needed in low-bandwidth communication environments.

The scheme is constructed from the Inverse Computational Diffie-Hellman Problem (Inv-CDHP) on bilinear pairings (see Section 1.2 below for a discussion of Inv-CDHP). The security of the scheme is based on the assumed hardness of this problem (which is widely accepted), which means there is no polynomial time algorithm to solve it with non-negligible probability. Bilinear pairings have been used to construct Identity (ID)-Based cryptosystems [B-F], so that the identity information of a user functions as his public key. The signing process in a short signature scheme can be regarded as the private key extract process in the ID-based public key setting from bilinear pairings. Therefore, the ZSS signature scheme can be regarded as being derived from Sakai-Kasahara's ID-based encryption scheme with pairing [S-K, RFC6508].

The algorithm is for use in the following context:

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

* where a message is to be signed and then verified (e.g., for authenticating the initiating party during key establishment);

* where a Certificate Authority (CA) or Trusted Third Party (TTP) within a traditional Public Key Infrastructure (PKI) provides a root of trust for both parties.

1.1 Bilinear Pairings

Let G_1 and G_2 be cyclic additive groups generated by P and P', respectively, both of whose order is a prime q. Let G_3 be a cyclic multiplicative group with the same order q. Let Z_q be the additive group of integers modulo q.

Let <,>: G_1 X G_2 --> G_3 be a map with the following properties.

1. Bilinearity: <aP,bQ>=<P,Q>^(ab) for all P, Q elements of G_1 and G_2, respectively, and a, b elements of Z_q.

2. Non-degeneracy: There exists P, Q elements of G_1 and G_2, respectively, such that <P,Q> != 1. In other words, the map does not send all pairs in G_1 X G_2 to the identity in G_3.

3. Computability: There is an efficient algorithm to compute <P,Q> for all P in G_1 and Q in G_2.

In our setting of prime order groups, non-degeneracy is equivalent to <P,Q> != 1 for all nontrivial P, Q elements in G_1 and G_2, respectively. So, when P is a generator of G_1 and Q is a generator of G_2, then <P,Q> is a generator of G_3. Such a bilinear map is called a bilinear pairing.

1.2 Discrete Logarithm Problem and Diffie-Hellman Problems

We consider the following problems in the additive group (G_1;+).

Discrete Logarithm Problem (DLP): Given two group elements P and Q, find an integer n in (Z_q)*, such that Q=nP whenever such an integer exists.

Decision Diffie-Hellman Problem (DDHP): For a,b,c in (Z_q)*, given P, aP, bP, cP decide whether c is congruent to ab mod q.

Computational Diffie-Hellman Problem (CDHP): For a,b in (Z_q)*,

given P, aP, bP, compute abP.

Inverse Computational Diffie-Hellman Problem (Inv-CDHP): For a in (Z_q)*, given P, aP, compute [a^(-1)]P.

Square Computational Diffie-Hellman Problem (Squ-CDHP): For a in (Z_q)*, given P, aP, compute [a^2]P.

Bilinear Diffie-Hellman problem (BDHP): Given (P, aP, bP, cP) for some a,b,c in (Z_q)*, compute v in G_3 such that v = <P,P>^(abc).

The CDHP, Inv-CDHP, and Squ-CDHP are polynomial time equivalent. The DLP, CDHP, Inv-CDHP, Squ-CDHP, and BDHP are assumed to be hard, which means there is no polynomial time algorithm to solve any of them with non-negligible probability. Therefore, the security of pairing based cryptosystems are typically based on these problems. A Gap Diffie-Hellman (GDH) group is a group in which the DDHP can be efficiently solved but the CDHP is intractable. The bilinear pairing gives us such a group, found on elliptic curves or hyperelliptic curves over finite fields. The bilinear pairings can be derived from the Weil or Tate pairing, as in [B-F, Cha-Cheon, Hess]. The ZSS scheme works on any GDH group, but in this document we focus on a particular family of ordinary (i.e., non-supersingular) elliptic curves, known as BN curves, described in Section 3.4 and the pairing described in Appendix A.2.

1.3 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 RFC 2119 [RFC2119].

2 Architecture

We consider the situation where one entity (the Signer) wishes to sign a message that it is sending to another entity (the Verifier).

As in a traditional Public Key Infrastructure (PKI), a Certificate Authority (CA) or Trusted Third Party (TTP) provides assurance of a signer's identity, which is bound to the signer's public key. The CA may generate a public key and private key (a key pair) or the signer may generate their own key pair and register the Signer Public Key (SPK) with a CA.

The mechanism by which a secret key is transported MUST be secure, as the security of the authentication provided by ZSS signatures is no stronger than the security of this supply channel. The choice of secret key transport mechanism is outside the scope of this document.

During the signing process, once the Signer has formed its message, it signs the message using its Signer Secret Key (SSK). It transmits the Signature with the message. The Verifier MUST then use the message, Signature, and SPK in verification.

This document specifies

* an algorithm for creating a Signature from a message, using an SSK;

* an algorithm for verifying a Signature for a message, using an SPK.

This document does not specify (but comments on)

* how to choose a valid and secure elliptic curve;

* which hash function to use.

3 Notation, Definitions and Parameters

3.1 Notation

n A security parameter; n should be at most half the bit size of q.

p A prime, of size at least 2n bits, which is the order of the finite field F_p. In this document, p is always congruent to 3 modulo 4.

F_p The finite field of order p (i.e., field with p elements).

F* The multiplicative group of the non-zero elements in the field F; e.g., (F_p)* is the multiplicative group of the finite field F_p.

q The order of E(F_p). In this document, for BN curves, q is always prime. To provide the desired level of security, lg(q) MUST be greater than 2*n.

E An elliptic curve defined over F_p, having prime order q. In this document, we use BN elliptic curves with equation y^2 = x^3 + 2 modulo p.

E' A sextic twist of the elliptic curve E. In this document, E':y^2 = x^3 + (1-i) over F_p^12. The order of E' over F_{p^2} is q(2p-q).

E(F) The additive group of points of affine coordinates (x,y) with x, y in the field F, that satisfy the curve equation for E.

P A generator of E(F_p). P has order q.

P' A point of E'(F_p^2) that generates the cyclic subgroup of order q.

0 The null element of any additive group of points on an elliptic curve, also called the point at infinity.

F_p^2 The extension field of degree 2 of the field F_p. In this document, we use a particular instantiation of this field; F_p^2 = F_p[i], where i^2 + 1 = 0. It is for this reason that we choose p congruent to 3 modulo 4.

G[q] The q-torsion of a group G. This is the subgroup generated by points of order q in G.

< , > The Ate pairing. In this document, this is a bilinear map from E'(F_p^2)[q] X E(F_p)[q] onto the subgroup of order q in (F_p^12)*. A full definition is given in Appendix A.2.

g g = <P,P'>. Having this pre-computed value allows the Verifier to only perform one pairing operation to verify a signature.

H A cryptographic hash function. [FIPS180-3] contains NIST approved hash functions.

lg(x) The base 2 logarithm of the real value x.

3.2 Definitions

Certificate Authority (CA) - The Certificate Authority is a trusted third party who provides assurance that the SPK belongs to the signer and verified proof of the signer's identity when the signer registered the SPK.

Public parameters - The public parameters are a set of parameters that are held by all users of the system. Each application of ZSS MUST define the set of public parameters to be used. The parameters needed are n, p, q, E, P, P', < , >, g, and H.

Signer Public Key (SPK) - The Signer's Public key is used to verify the signature of the entity whose SSK corresponds to the SPK. It is a point on the elliptic curve E.

Signer Secret Key (SSK) - The Signer's Secret Key is used to generate a signature and must not be revealed to any entity other than the trusted third party and the authorized signer. It is a value between 2 and q-1.

3.3 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.

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].

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. Such octet strings MUST have length L = Ceiling(lg(p)/8).

F_p^2 elements — The elements of F_p^2 = F_p[i] are represented as x_1 + i * x_2, where x_1 and x_2 are elements of F_p. It is for this reason that we choose p congruent to 3 modulo 4.

Points on E, E' — Elliptic curve points MUST be represented in uncompressed form 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 octet string representing x, y' is the octet string representing y, and || denotes concatenation. The representation is 2*L+1 octets in length.

3.4 Arithmetic

ZSS relies on elliptic curve arithmetic. The coordinates of a point P on the elliptic curve are given by P = (P_x,P_y), where Px and Py are the affine coordinates in F_p satisfying the curve equation.

The following conventions are assumed for curve operations:

Point addition - If P and Q are two points on a curve E, their sum is denoted as P + Q.

Scalar multiplication - If P is a point on a curve, and k an integer, the result of adding P to itself a total of k times is denoted [k]P.

In this document, we use BN curves with equation y^2 = x^3 + 2 modulo p. This curve is chosen because of the many efficiency and simplicity advantages it offers, as mentioned in Section 1 and discussed in [Pereira]. For example, one advantage is an easy determination of a generator P of E(F_p), namely P = (-1,1).

4 The ZSS Cryptosystem

This section describes the ZSS short signature scheme [ZSS].

4.1 Parameter Generation

The following static parameters are fixed for each implementation. They are not intended to change frequently, and MUST be specified for each user community.

The system parameters to be generated for a given security parameter n are {p, q, E, P, P', <,>, g, H}. These are known by the Sender and the Verifier.

4.2 Key Generation

To create signatures, each Signer requires an SSK and SPK. The SSK is an integer, and the SPK is an elliptic curve point. The SSK MUST be kept secret (to the Signer and possibly the CA), but the SPK need not be kept secret.

The Signer (or CA) MUST randomly select a value in the range 2 to q-1, and assigns this value to x, which is the SSK.

The Signer MUST derive its SPK, X, by performing the calculation X =[x]P.

If the signer generated the SPK, then it must be registered with a CA.

4.3 Signature Generation

Given the SSK x, and a message m, the Signer computes the signature S by performing the following steps:

1) Compute the hash of the message as a mod q value using the hash algorithm specified in the public parameters.

2) Compute (H(m)+x)^-1, where the inversion is performed modulo q.

3) Compute S = [(H(m)+x)^-1]P'. The signature is S, and this is a point on the curve E'.

The Signer sends m and S.

4.4 Signature Verification

Given the SPK X, a message m, and a signature S, the Receiver verifies that <[H(m)]P + X, S> = g, to ensure that the Signer is authentic and the message was not altered in transit. This is achieved by the Verifier performing the following steps:

1) Check that S is a point on the curve E', otherwise reject the signature.

2) Compute the hash of the message as a mod q value using the hash algorithm specified in the public parameters.

3) Compute the elliptic curve point [H(m)]P + X.

4) Compute the pairing <[H(m)]P + X, S>.

5) Verify that <[H(m)]P + X, S> = g; if not, reject the signature.

5 Security Considerations

This document describes the ZSS Short Signature Scheme. We assume that the security provided by this algorithm depends entirely on the secrecy of the secret keys it uses, and that for an adversary to defeat this security, he will need to perform computationally intensive cryptanalytic attacks to recover a secret key. Note that a security proof exists for ZSS in the Random Oracle Model [ZSS].

When defining public parameters, guidance on parameter sizes from [RFC4492] SHOULD be followed. For lower security levels (e.g., less than 128 bit security), the parameter sizes must be determined based on the elliptic curve discrete logarithm problem over F_p, and for the higher security levels the parameter sizes are based on the finite field size (e.g., 12*lg(p)). Table 1 shows bits of security afforded by various sizes of p.

Security (bits)	EC size (lg(p)	finite field size (12*lg(p))

80	160	1920
112	224	2688
128	256	3072
192	640	7680
256	1280	15360

Table 1: Comparable Strengths, taken from [RFC4492]

The order of the base point P used in ZSS, and hence the order of E(F_p) for BN curves, MUST be a large prime q. If n bits of security are needed, then lg(q) SHOULD be chosen to be at least 2*n. Similarly, if n bits of security are needed, then a hash with output size at least 2*n SHOULD be chosen.

Randomizing the messages that are signed is a way to enhance the security of the cryptographic hash function. [SP800-106] provides a technique to randomize messages that are input to a cryptographic hash function during the signature generation step. The intent of this method is to strengthen the collision resistance provided by the hash functions without any changes to the core hash functions and signature algorithms. If the message is randomized with a different random value each time it is signed, it will result in the message having a different digital signature each time.

Each user's SSK protects the ZSS communications it receives. This key MUST NOT be revealed to any entity other than the authorized user and possibly the CA (if the CA generated the key pair).

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

The randomness of values stipulated to be selected at random, as described in this document, is essential to the security provided by ZSS. If the value of x 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].

6 IANA Considerations

This memo includes no request to IANA.

7 References

7.1 Normative References

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

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.

[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.

[ZSS] Zhang, F., Safavi-Naini, R., and Susilo, W., "An Efficient Signature Scheme from Bilinear Pairings and Its Applications", PKC 2004, LNCS 2947, Springer-Verlag (2004), pp. 277-290.

7.2 Informative References

[B-F] Boneh, D., Franklin, M., "Identity-based encryption from the Weil pairing", Advances in Cryptology - Crypto 2001, LNCS 2139, Springer-Verlag (2001), pp. 213-229.

[Cha-Cheon] Cha, J.C., Cheon, J.H., "An identity-based signature from gap Diffie-Hellman groups", Public Key Cryptography - PKC 2003, LNCS 2139, Springer-Verlag (2003), pp. 18-3.

[Devegili] Devegili, A.J., Scott, M., Dahab, R., "Implementing Cryptographic Pairings over Barreto-Naehrig Curves", Pairing 2007, 197-207.

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

[Hess] Hess, F., "Efficient identity based signature schemes based on pairings", SAC 2002, LNCS 2595, Springer-Verlag (2002), pp. 310-324.

[Miller] Miller, V., "The Weil pairing, and its efficient

calculation", J. Cryptology 17 (2004), 235-261.

[P1363] IEEE P1363-2000, "Standard Specifications for Public-Key Cryptography", 2001.

[Pereira] Pereira, G. C., et al. "A Family of Implementation-Friendly BN Elliptic Curves", J. Systems and Software, Volume 84, Issue 8, Elsevier (2011), pp 1319-1326.

[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.

[S-K] Sakai, R., Ohgishi, K., and M. Kasahara, "ID based cryptosystem based on pairing on elliptic curves", Symposium on Cryptography and Information Security - SCIS, 2001.

[SP800-106] Dang, Q., "Randomized Hashing for Digital Signatures", NIST Special Publication 800-106, February 2009.

Appendix A. BN Elliptic Curves, Twists, Pairings and Supporting Algorithms

A.1. BN Elliptic Curves

E is an ordinary elliptic curve, known as a BN curve (of j-invariant 0), where E: y^2=x^3+b, defined over a finite prime field F_p. In this document, we let b = 2. We require that p is congruent to 3 modulo 4, for efficiency reasons. E has prime order q = #E(F_p), and for BN curves, the primes p and q are given by p = p(u) = 36u^4+36u^3+24u^2+6u+1 and q = q(u) = 36u^4+36u^3+18u^2+6u+1, for some integer u. The curve in this document has a generator P = (-1,1). BN curves have embedding degree k = 12 and admit a sextic twist, which allows for an optimal ate pairing on the groups, as we discuss below.

Routines for point addition and doubling on E(F_p) can be found in Appendix A.10 of [P1363].

A.2. Sextic Twists Since p is a prime congruent to 3 modulo 4, the

finite field F_p^2 can be represented as F_p[i]/(i^2+1). So i^2+1 = 0 and elements of F_p^2 are represented as x_1 + i * x_2, where x_1 and x_2 are elements of F_p. We may view F_p^12 as F_p^2[x]/(x^6-z), where x^6-z is irreducible over F_p^2.

Consider the twisting isomorphism, psi: E'(F_p^2) --> E(F_p), where (x',y') is mapped to (x'z^2),y'z^3) for some z in the multiplicative group of F_p^12. It can be shown that E':y^2 = x^3 +b/z over F_p^2, where z is not a cube nor square in F_p^2. E' is called the sextic twist of E over F_p^2. E'(F_p^2)[q] has a generator P' = [h](-i,1) where h=2p-q. So in the case of E: y^2=x^3+2 over F_p, we have E': y^2=x^3+(1-i) over F_p^2.

A.3. The Ate Pairing

The Tate, Ate or R-ate pairings can be used with BN curves in ZSS, but we describe the Ate pairing in this document The Ate pairing for BN curves uses roughly half the number of iterations of the Miller loop needed to compute the Tate pairing.

In general, the Ate pairing is from G_2 X G1 onto the subgroup of order q in (F_p^12)*, where G_2 = E(F_p^12)[q] and G_1 = E(F_p)[q]. Thus, the Ate pairing <Q,R> takes a point Q in E(F_p^12) and a point R in E(F_p), and evaluates f_Q(R), where f_Q is some polynomial over F_p^12 whose divisor is (q)(Q) - (q)(O). (Note that f_Q is defined only up to scalars of F_p^12.) Miller's algorithm [Miller] provides a method for evaluation of f_Q(R).

However, for BN curves, instead of using the full point Q in

E(F_p^12), we can use Q' in E'(F_p^2), where E' is the twist under the twisting isomorphism described in the section above, so psi(Q')=Q. This allows us to use a compact representation of the point and to avoid F_p^12 arithmetic when computing the pairing.

Thus, let us consider G_1 = E(F_p)[q] and G_2 = E'(F_p^2)[q]. We note that if Q=(Q_x,Q_y) and Q'=(Q_x',Q_y'), then (Q_x,Q_y)= ((z^2)Q_x',(z^3)Q_y'). The version of the Ate pairing used in this document is given by <Q',R> = f_Q'(R)^c in (F_p^12)*, where c=(p^12-1)/q. It satisfies the bilinear relation <[x]Q',R> = <Q',[x]R> = <Q',R>^x for all Q' in E'(F_p^2)[q] and R in E(F_p)[q], for all integers x.

We provide pseudocode for computing <Q',R> with elliptic curve arithmetic expressed in affine coordinates. From this point forward, we will drop the notation of Q' and just use Q, understanding that Q is a point on E'(F_p^2). Note that this section does not fully describe the most efficient way of computing the pairing, as there are further ways of reducing the number and complexity of the operations needed to compute the pairing (e.g., [Devegili]). For example, a common optimization is to factor c = (p^12-1)/q into three parts: (p^6-1), (p^2+1) and (p^4-p^2+1)/q.

```
<CODE BEGINS>

/* Copyright (c) 2012 IETF Trust and the persons identified as
authors of the code.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, is permitted pursuant to, and subject to the license
terms contained in, the Simplified BSD License set forth in Section
4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info). */

     Routine for computing the pairing <Q,R>:

       Input Q, a point in E'(F_p^2)[q], and R, a point on
       E(F_p)[q].

       Initialize variables:
          f = (F_p^12)*;    // An element of (F_p^12)*
          C = Q;         // An element of E'(F_p^2)[q]
          c = (p^12-1)/q;   // An integer

       for bits of q-1, starting with the second most significant
       bit, ending with the least significant bit, do

          // gradient of line through C, C, [-2]C.
```

```
               l = 3*( C_x^2 ) / ( 2*C_y );

               //accumulate line evaluated at R into f

               f = f^2 * ( l*( - R_x + C_x ) + ( R_y - C_y ) );

               C = [2]C;

               if bit is 1, then

                  // gradient of line through C, Q, -C-Q.

                  l = ( C_y - Q_y )/( C_x - Q_x );

                  //accumulate line evaluated at R into f

                  f = f * ( l*( - R_x + C_x ) + ( R_y - C_y ) );

                  C = C+Q;

                end if;

              end for;

              t = f^c;

              return representative in (F_p^12)* of t;

      <CODE ENDS>
```

A.4. Hashing to an Integer Range

We use the function HashToIntegerRange(s, n, hashfn) to hash strings to an integer range. Given a string (s), a hash function (hashfn), and an integer (n), this function returns a value between 0 and n - 1.

Input:

* an octet string, s

* an integer, n <= (2^hashlen)^hashlen

* a hash function, hashfn, with output length hashlen bits

Output:

* an integer, v, in the range 0 to n-1

Method:

1) Let A = hashfn(s)

2) Let h_0 = 00...00, a string of null bits of length hashlen bits

3) Let l = Ceiling(lg(n)/hashlen)

4) For each i in 1 to l, do:

   a) Let h_i = hashfn(h_(i - 1))

   b) Let v_i = hashfn(h_i || A), where || denotes concatenation

5) Let v' = v_1 || ... || v_l

6) Let v = v' mod n

Appendix B. Example Data

This appendix provides example data for the ZSS short signature scheme with the public parameters (n, p, q, E, P, P', g, H).

We denote elements of Fp_2 by (alpha, beta) for alpha + i*beta, where i in Fp_2 is a root of X^2+1. We denote elements of Fp_12 by ((gamma_0), (gamma_1), (gamma_2), (gamma_3), (gamma_4), (gamma_5)) for gamma_0 + gamma_1*Z + gamma_2*Z^2 + gamma_3*Z^3 + gamma_4*Z^4 + gamma_5*Z^5, where Z in Fp_12 is a root of x^6-z and gamma_j=(alpha_j, beta_j) are elements of Fp_2.

B.1 Example 1

```
n = 111 and lg(p) = 222

p = p(u) = p(18577485901856771)
  = 4287955428141231242568449143434307004509408777842990443037248720923

q = q(u) = q(18577485901856771)
  = 4287955428141231242568449143434304933771514175720485558004857442277

E: y^2 = x^3 + 2

   Thus, E': y'^2 = x'^3 + (1, 4287955428141231242568449143434307004509408777842990443037248720922)
```

```
P = (-1,1)
  = (4287955428141231242568449143434307004509408777842990443037248720
  922, 1)

P' = [h](-i,1)=(P'x,P'y) where P'x and P'y are elements of Fp_2 and

    P'x = (4150062559496717098010848775383950942405168339753729033708
    91183040, 3201547425391805499092396849287316949217764330692173661
    045966218196)

    P'y = (2739493618867793153898969775521693478241455091691296751947
    733115593, 1664707679450242347152392883299791673446597080006587338
    7257740360355)

g = <P, P'> is an element of Fp_12 given by

  ((36699339660626669649427399393560215125019796922092031025948063026
  15, 247927466780520015848965517635750695647326440402643111951301234
  2995),

  (53440222253683237877269791416098727648815280997704796178179123615
  5, 152660447468210228523508984531632103922262052241800765770764185
  1816),

  (49145419234025334524484519757289516582314844784075278041512123443,
  176265594691520614080041515255534401194732308979032729979138040295
  4),

  (23696703075178036967923154124988633199564384190798395748600577760
  85, 23618404356311472082427998195066752928773113611373491721529464
  37996),

  (12762283628400824508803470970369296405340301501303861200304473348
  06, 13459347095597820798563091367816489138601686487822582178268889
  4443),

  (52536691068804792250953154403977106032469644538529906910544187717
  3, 111049479509897086258622292367384411180467475476136042303440990
  5476))

SSK = 2121608753564392499593333521375987220574081909435960440370410
821656

SPK = (120851890594243637869885901573990997912577177623964284017952
0609651, 5156510979532175335881708985883359220634082111209944466019
22864253)

Suppose H(m) = 6104193801232612202724894323091424875875271378342228
```

```
   81137528506355

   Signature S = (Sx,Sy) where Sx and Sy are elements of Fp_2 and

   Sx = (460397035846801088503894950670120132397930040162387325467414 3
   95606, 3901848544113768795823081213318287160380301464573351188507 88
   9233137)

   Sy = (294244928255842354933084300809365734367769683929211592236129 7
   49298, 2026256877874569973858226679691650946337449890951287636000 03
   736610)

   For verification of the signature:
          <H(m)P + X, S> = g
```

B.2 Example 2

```
   n = 127 and lg(p) = 254

   p = p(u) = p(-4647714815446351873)
     = 1679810873101583228494080414223173390988918712143906984893371542
       6072753864723

   q = q(u) = q(-4647714815446351873)
     = 16798108731015832284940804142231733909759579603404752749028378864
       165570215949

   E: y^2 = x^3 + 2

      Thus, E': y'^2 = x'^3 + (1, 1679810873101583228494080414223173390 9
      889187121439069848933715426072753864722)

   P = (-1,1)
     = (1679810873101583228494080414223173390988918712143906984893371542
       6072753864722, 1)

   P' = [h](-i,1) = (P'x,P'y), where P'x and P'y are elements of Fp_2

       P'x =
       (2759930593230997547690248631365636073479225314645471320757910281
       674905877291, 2301614907882718573745244110620256732212332571700 73
       76035129070751203315745 15)

       P'y =
       (9480765153516887970576068394945041092622478388406602889697250323
       02618946458, 666307744692739207922404563142529103669240282380266 3
       947112913140121004068507)
```

```
g = <P, P'> is an element of Fp_12 given by
  ((13070690801249658484759892809227642840919015841299984602661540278
  97835831306, 362837632692008901334341187262873478716707643732273036
  691371364602390573150 3),

  (35277875384519058374069094101471040868126180606524783772942203899 7
  7928485580, 139084204959536988114903704041505075186145820309773968 8
  07976269403163053627 87),

  (14895739131823503897972138357591096297360268227609321098943152635 1
  38088456200, 154193402372256829285477206567013233448625527219699948
  30027125771243100988775),

  (65701534525096536324405839594768633146749459533060058166986154590 9
  8579995196, 924632872007155968845772060705321833088964729559013933 8
  23862417580822596279 5),

  (15101466540660239552845468082274401614780748403849519674069680403 4
  7117671512, 696423195106307532437867295533009104570830155611345537 9
  31696775414877400453 0),

  (13200196240779235573717726113916392263745499355984208510745183366 3
  5435672354, 947633516865877259404557047678407354227586638702918931 7
  56020395954987665658 2))

SSK = 228064033978937665992889984775405287134161793365057496448735949
2611

SPK = (48893896735870064320433171153400539525040538030176968340812183
01282547698392, 15356945755932217528217084848811599775130985825038998
69296524319810590462444 2)

Suppose H(m) =  21668398097129279358779433271119370918865051659048528
91187078055077

Signature S = (Sx,Sy) where Sx and Sy are elements of Fp_2 and

Sx = (72905198149775047301898989459265776974343781845977477556122490 0
9723218090232, 683378059974468691645078542720737033649767207447427118
64727097976181206516 15)

Sy = (15743217482738606986081218493139987785782632881737317277126416 6
63269695635786, 93427588866953969700345687463198658107209055412980315
33851535785638159753756)

For verification of the signature:
       <H(m)P + X, S> = g
```

Author's Address

Laura Hitt
6011 W Courtyard Dr.
Building 5, Suite 300
Austin, TX 78730

EMail: Lhitt@21CT.com