

Internet Engineering Task Force (IETF)  
Request for Comments: 6508  
Category: Informational  
ISSN: 2070-1721

M. Groves  
CESG  
February 2012

## Sakai-Kasahara Key Encryption (SAKKE)

### Abstract

In this document, the Sakai-Kasahara Key Encryption (SAKKE) algorithm is described. This uses Identity-Based Encryption to exchange a shared secret from a Sender to a Receiver.

### Status of This Memo

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

This document is a product of the Internet Engineering Task Force (IETF). It has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

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/rfc6508>.

### Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction .....	2
1.1. Requirements Terminology .....	3
2. Notation and Definitions .....	3
2.1. Notation .....	3
2.2. Definitions .....	5
2.3. Parameters to Be Defined or Negotiated .....	6
3. Elliptic Curves and Pairings .....	7
3.1. $E(F_p^2)$ and the Distortion Map .....	7
3.2. The Tate-Lichtenbaum Pairing .....	7
4. Representation of Values .....	9
5. Supporting Algorithms .....	10
5.1. Hashing to an Integer Range .....	10
6. The SAKKE Cryptosystem .....	11
6.1. Setup .....	11
6.1.1. Secret Key Extraction .....	11
6.1.2. User Provisioning .....	11
6.2. Key Exchange .....	12
6.2.1. Sender .....	12
6.2.2. Receiver .....	12
6.3. Group Communications .....	13
7. Security Considerations .....	13
8. References .....	15
8.1. Normative References .....	15
8.2. Informative References .....	15
Appendix A. Test Data.....	17

## 1. Introduction

This document defines an efficient use of Identity-Based Encryption (IBE) based on bilinear pairings. The Sakai-Kasahara IBE cryptosystem [S-K] is described for establishment of a shared secret value. This document adds to the IBE options available in [RFC5091], providing an efficient primitive and an additional family of curves.

This document is restricted to a particular family of curves (see Section 2.1) that have the benefit of a simple and efficient method of calculating the pairing on which the Sakai-Kasahara IBE cryptosystem is based.

IBE schemes allow public and private keys to be derived from Identifiers. In fact, the Identifier can itself be viewed as corresponding to a public key or certificate in a traditional public key system. However, in IBE, the Identifier can be formed by both Sender and Receiver, which obviates the necessity of providing public keys through a third party or of transmitting certified public keys

during each session establishment. Furthermore, in an IBE system, calculation of keys can occur as needed, and indeed, messages can be sent to users who are yet to enroll.

The Sakai-Kasahara primitive described in this document supports simplex transmission of messages from a Sender to a Receiver. The choice of elliptic curve pairing on which the primitive is based allows simple and efficient implementations.

The Sakai-Kasahara Key Encryption scheme described in this document is drawn from the Sakai-Kasahara Key Encapsulation Mechanism (SK-KEM) scheme (as modified to support multi-party communications) submitted to the IEEE P1363 Working Group in [SK-KEM].

### 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. Notation and Definitions

### 2.1. Notation

- $n$  A security parameter; the size of symmetric keys in bits to be exchanged by SAKKE.
- $p$  A prime, 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  $p$ .
- $F^*$  The multiplicative group of the non-zero elements in the field  $F$ ; e.g.,  $(F_p)^*$  is the multiplicative group of the finite field  $F_p$ .
- $q$  An odd prime that divides  $p + 1$ . To provide the desired level of security,  $\lg(q)$  MUST be greater than  $2*n$ .
- $E$  An elliptic curve defined over  $F_p$ , having a subgroup of order  $q$ . In this document, we use supersingular curves with equation  $y^2 = x^3 - 3 * x$  modulo  $p$ . This curve is chosen because of the efficiency and simplicity advantages it offers. The choice of  $-3$  for the coefficient of  $x$  provides advantages for elliptic curve arithmetic that are explained in [P1363]. A further reason for this choice of curve is that Barreto's trick [Barreto] of eliminating the computation of the denominators when calculating the pairing applies.

- $E(F)$  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 point of  $E(F_p)$  that generates the cyclic subgroup of order  $q$ . The coordinates of  $P$  are given by  $P = (P_x, P_y)$ . These coordinates are in  $F_p$ , and they satisfy the curve equation.
- $0$  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  $i^2 + 1 = 0$ .
- $PF_p$  The projectivization of  $F_p$ . We define this to be  $(F_p^2)^*/(F_p)^*$ . Note that  $PF_p$  is cyclic and has order  $p + 1$ , which is divisible by  $q$ .
- $G[q]$  The  $q$ -torsion of a group  $G$ . This is the subgroup generated by points of order  $q$  in  $G$ .
- $\langle , \rangle$  A version of the Tate-Lichtenbaum pairing. In this document, this is a bilinear map from  $E(F_p)[q] \times E(F_p)[q]$  onto the subgroup of order  $q$  in  $PF_p$ . A full definition is given in Section 3.2.
- Hash A cryptographic hash function.
- $\lg(x)$  The base 2 logarithm of the real value  $x$ .

The following conventions are assumed for curve operations:

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

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

We assume that the following concrete representations of mathematical objects are used:

Elements of  $F_p$  - The  $p$  elements of  $F_p$  are represented directly using the integers from 0 to  $p-1$ .

Elements of  $F_p^2$  - 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  $F_p$ .

Elements of  $PF_p$  - Elements of  $PF_p$  are cosets of  $(F_p)^*$  in  $(F_p^2)^*$ . Every element of  $F_p^2$  can be written unambiguously in the form  $x_1 + i * x_2$ , where  $x_1$  and  $x_2$  are elements of  $F_p$ . Thus, elements of  $PF_p$  (except the unique element of order 2) can be represented unambiguously by  $x_2/x_1$  in  $F_p$ . Since  $q$  is odd, every element of  $PF_p[q]$  can be represented by an element of  $F_p$  in this manner.

This representation of elements in  $PF_p[q]$  allows efficient implementation of  $PF_p[q]$  group operations, as these can be defined using arithmetic in  $F_p$ . If  $a$  and  $b$  are elements of  $F_p$  representing elements  $A$  and  $B$  of  $PF_p[q]$ , respectively, then  $A * B$  in  $PF_p[q]$  is represented by  $(a + b)/(1 - a * b)$  in  $F_p$ .

## 2.2. Definitions

Identifier - Each user of an IBE system MUST have a unique, unambiguous identifying string that can be easily derived by all valid communicants. This string is the user's Identifier. An Identifier is an integer in the range 2 to  $q-1$ . The method by which Identifiers are formed MUST be defined for each application.

Key Management Service (KMS) - The Key Management Service is a trusted third party for the IBE system. It derives system secrets and distributes key material to those authorized to obtain it. Applications MAY support mutual communication between the users of multiple KMSs. We denote KMSs by  $KMS_T$ ,  $KMS_S$ , etc.

Public parameters - The public parameters are a set of parameters that are held by all users of an IBE system. Such a system MAY contain multiple KMSs. Each application of SAKKE MUST define the set of public parameters to be used. The parameters needed are  $p$ ,  $q$ ,  $E$ ,  $P$ ,  $g = \langle P, P \rangle$ , Hash, and  $n$ .

Master Secret ( $z_T$ ) - The Master Secret  $z_T$  is the master key generated and privately kept by  $KMS_T$  and is used by  $KMS_T$  to generate the private keys of the users that it provisions; it is an integer in the range 2 to  $q-1$ .

KMS Public Key:  $Z_T = [z_T]P$  - The KMS Public Key  $Z_T$  is used to form Public Key Establishment Keys for all users provisioned by  $KMS_T$ ; it is a point of order  $q$  in  $E(F_p)$ . It MUST be provisioned by  $KMS_T$  to all who are authorized to send messages to users of the IBE system.

Receiver Secret Key (RSK) - Each user enrolled in an IBE system is provisioned with a Receiver Secret Key by its KMS. The RSK provided to a user with Identifier 'a' by KMS\_T is denoted  $K_{(a,T)}$ . In SAKKE, the RSK is a point of order  $q$  in  $E(F_p)$ .

Shared Secret Value (SSV) - The aim of the SAKKE scheme is for the Sender to securely transmit a shared secret value to the Receiver. The SSV is an integer in the range  $0$  to  $(2^n) - 1$ .

Encapsulated Data - The Encapsulated Data are used to transmit secret information securely to the Receiver. They can be computed directly from the Receiver's Identifier, the public parameters, the KMS Public Key, and the SSV to be transmitted. In SAKKE, the Encapsulated Data are a point of order  $q$  in  $E(F_p)$  and an integer in the range  $0$  to  $(2^n) - 1$ . They are formatted as described in Section 4.

### 2.3. Parameters to Be Defined or Negotiated

In order for an application to make use of the SAKKE algorithm, the communicating hosts MUST agree on values for several of the parameters described above. The curve equation ( $E$ ) and the pairing ( $\langle , \rangle$ ) are constant and used for all applications.

For the following parameters, each application MUST either define an application-specific constant value or define a mechanism for hosts to negotiate a value:

- \*  $n$
- \*  $p$
- \*  $q$
- \*  $P = (P_x, P_y)$
- \*  $g = \langle P, P \rangle$
- \* Hash

### 3. Elliptic Curves and Pairings

$E$  is a supersingular elliptic curve (of  $j$ -invariant 1728).  $E(F_p)$  contains a cyclic subgroup of order  $q$ , denoted  $E(F_p)[q]$ , whereas the larger object  $E(F_{p^2})$  contains the direct product of two cyclic subgroups of order  $q$ , denoted  $E(F_{p^2})[q]$ .

$P$  is a generator of  $E(F_p)[q]$ . It is specified by the (affine) coordinates  $(P_x, P_y)$  in  $F_p$ , satisfying the curve equation.

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

#### 3.1. $E(F_{p^2})$ and the Distortion Map

If  $(Q_x, Q_y)$  are (affine) coordinates in  $F_p$  for some point (denoted  $Q$ ) on  $E(F_p)[q]$ , then  $(-Q_x, iQ_y)$  are (affine) coordinates in  $F_{p^2}$  for some point on  $E(F_{p^2})[q]$ . This latter point is denoted  $[i]Q$ , by analogy with the definition for scalar multiplication. The two points  $P$  and  $[i]P$  together generate  $E(F_{p^2})[q]$ . The map  $[i]: E(F_p) \rightarrow E(F_{p^2})$  is sometimes termed the distortion map.

#### 3.2. The Tate-Lichtenbaum Pairing

We proceed to describe the pairing  $\langle \cdot, \cdot \rangle$  to be used in SAKKE. We will need to evaluate polynomials  $f_R$  that depend on points on  $E(F_p)[q]$ . Miller's algorithm [Miller] provides a method for evaluation of  $f_R(X)$ , where  $X$  is some element of  $E(F_{p^2})[q]$  and  $R$  is some element of  $E(F_p)[q]$  and  $f_R$  is some polynomial over  $F_p$  whose divisor is  $(q)(R) - (q)(0)$ . Note that  $f_R$  is defined only up to scalars of  $F_p$ .

The version of the Tate-Lichtenbaum pairing used in this document is given by  $\langle R, Q \rangle = f_R([i]Q)^c / (F_p)^*$ . It satisfies the bilinear relation  $\langle [x]R, Q \rangle = \langle R, [x]Q \rangle = \langle R, Q \rangle^x$  for all  $Q, R$  in  $E(F_p)[q]$ , for all integers  $x$ . Note that the domain of definition is restricted to  $E(F_p)[q] \times E(F_p)[q]$  so that certain optimizations are natural.

We provide pseudocode for computing  $\langle R, Q \rangle$ , with elliptic curve arithmetic expressed in affine coordinates. We make use of Barreto's trick [Barreto] for avoiding the calculation of denominators. Note that this section does not fully describe the most efficient way of computing the pairing; it is possible to compute the pairing without any explicit reference to the extension field  $F_{p^2}$ . This reduces the number and complexity of the operations needed to compute the pairing.

<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 <R,Q>:

Input R, Q points on E(F<sub>p</sub>)[q];

Initialize variables:

```
v = (F_p)*;    // An element of PF_p[q]
C = R;        // An element of E(F_p)[q]
c = (p+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 - 1 ) / ( 2*C_y );
```

```
//accumulate line evaluated at [i]Q into v
v = v^2 * ( l*( Q_x + C_x ) + ( i*Q_y - C_y ) );
```

C = [2]C;

if bit is 1, then

```
// gradient of line through C, R, -C-R.
l = ( C_y - R_y ) / ( C_x - R_x );
```

```
//accumulate line evaluated at [i]Q into v
v = v * ( l*( Q_x + C_x ) + ( i*Q_y - C_y ) );
```

C = C+R;

end if;  
end for;

t = v^c;

```
return representative in  $F_p$  of  $t$ ;
```

```
End of routine;
```

```
Routine for computing representative in  $F_p$  of elements of  $PF_p$ :
```

```
Input  $t$ , in  $F_p^2$ , representing an element of  $PF_p$ ;
```

```
Represent  $t$  as  $a + i*b$ , with  $a, b$  in  $F_p$ ;  
return  $b/a$ ;
```

```
End of routine;
```

```
<CODE ENDS>
```

#### 4. Representation of Values

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.

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 $L = \text{Ceiling}(\lg(p)/8)$ .
$PF_p$ elements	Elements of $PF_p$ MUST be represented as an element of $F_p$ using the algorithm in Section 3.2. They are therefore represented as octet strings as defined above and are $L$ octets in length. Representation of the unique element of order 2 in $PF_p$ will not be required.
Points on $E$	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.

**Encapsulated Data** The Encapsulated Data MUST be represented as an elliptic curve point concatenated with an integer in the range 0 to  $(2^n) - 1$ . Since the length of the representation of elements of  $F_p$  is well defined given  $p$ , these data can be unambiguously parsed to retrieve their components. The Encapsulated Data is  $2*L + n + 1$  octets in length.

## 5. Supporting Algorithms

### 5.1. 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,  $n \leq (2^{\text{hashlen}})^{\text{hashlen}}$
- \* a hash function, `hashfn`, with output length `hashlen` bits

Output:

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

Method:

- 1) Let  $A = \text{hashfn}( s )$
- 2) Let  $h_0 = 00\dots00$ , a string of null bits of length `hashlen` bits
- 3) Let  $l = \text{Ceiling}(\lg(n)/\text{hashlen})$
- 4) For each  $i$  in 1 to  $l$ , do:
  - a) Let  $h_i = \text{hashfn}(h_{(i - 1)})$
  - b) Let  $v_i = \text{hashfn}(h_i || A)$ , where  $||$  denotes concatenation

5) Let  $v' = v_1 || \dots || v_l$

6) Let  $v = v' \bmod n$

## 6. The SAKKE Cryptosystem

This section describes the Sakai-Kasahara Key Encryption algorithm. It draws from the cryptosystem first described in [S-K].

### 6.1. Setup

All users share a set of public parameters with a KMS. In most circumstances, it is expected that a system will only use a single KMS. However, it is possible for users provisioned by different KMSs to interoperate, provided that they use a common set of public parameters and that they each possess the necessary KMS Public Keys. In order to facilitate this interoperation, it is anticipated that parameters will be published in application-specific standards.

KMS\_T chooses its KMS Master Secret,  $z_T$ . It MUST randomly select a value in the range 2 to  $q-1$ , and assigns this value to  $z_T$ . It MUST derive its KMS Public Key,  $Z_T$ , by performing the calculation  $Z_T = [z_T]P$ .

#### 6.1.1. Secret Key Extraction

The KMS derives each RSK from an Identifier and its KMS Master Secret. It MUST derive a RSK for each user that it provisions.

For Identifier 'a', the RSK  $K(a,T)$  provided by KMS\_T MUST be derived by KMS\_T as  $K(a,T) = [(a + z_T)^{-1}]P$ , where 'a' is interpreted as an integer, and the inversion is performed modulo  $q$ .

#### 6.1.2. User Provisioning

The KMS MUST provide its KMS Public Key to all users through an authenticated channel. RSKs MUST be supplied to all users through a channel that provides confidentiality and mutual authentication. The mechanisms that provide security for these channels are beyond the scope of this document: they are application specific.

Upon receipt of key material, each user MUST verify its RSK. For Identifier 'a', RSKs from KMS\_T are verified by checking that the following equation holds:  $\langle [a]P + Z, K(a,T) \rangle = g$ , where 'a' is interpreted as an integer.

## 6.2. Key Exchange

A Sender forms Encapsulated Data and sends it to the Receiver, who processes it. The result is a shared secret that can be used as keying material for securing further communications. We denote the Sender A with Identifier 'a'; we denote the Receiver B with Identifier 'b'; Identifiers are to be interpreted as integers in the algorithms below. Let A be provisioned by KMS\_T and B be provisioned by KMS\_S.

### 6.2.1. Sender

In order to form Encapsulated Data to send to device B who is provisioned by KMS\_S, A needs to hold Z\_S. It is anticipated that this will have been provided to A by KMS\_T along with its User Private Keys. The Sender MUST carry out the following steps:

- 1) Select a random ephemeral integer value for the SSV in the range 0 to  $2^n - 1$ ;
- 2) Compute  $r = \text{HashToIntegerRange}(\text{SSV} \parallel b, q, \text{Hash})$ ;
- 3) Compute  $R_{(b,S)} = [r]([b]P + Z_S)$  in  $E(F_p)$ ;
- 4) Compute the Hint, H;
  - a) Compute  $g^r$ . Note that  $g$  is an element of  $PF_p[q]$  represented by an element of  $F_p$ . Thus, in order to calculate  $g^r$ , the operation defined in Section 2.1 for calculation of  $A * B$  in  $PF_p[q]$  is to be used as part of a square and multiply (or similar) exponentiation algorithm, rather than the regular  $F_p$  operations;
  - b) Compute  $H := \text{SSV XOR HashToIntegerRange}(g^r, 2^n, \text{Hash})$ ;
- 5) Form the Encapsulated Data (  $R_{(b,S)}, H$  ), and transmit it to B;
- 6) Output SSV for use to derive key material for the application to be keyed.

### 6.2.2. Receiver

Device B receives Encapsulated Data from device A. In order to process this, it requires its RSK,  $K_{(b,S)}$ , which will have been provisioned in advance by KMS\_S. The method by which keys are provisioned by the KMS is application specific. The Receiver MUST carry out the following steps to derive and verify the SSV:

- 1) Parse the Encapsulated Data (  $R_{(b,S)}$ ,  $H$  ), and extract  $R_{(b,S)}$  and  $H$ ;
- 2) Compute  $w := \langle R_{(b,S)}, K_{(b,S)} \rangle$ . Note that by bilinearity,  $w = g^r$ ;
- 3) Compute  $SSV = H \text{ XOR } \text{HashToIntegerRange}( w, 2^n, \text{Hash} )$ ;
- 4) Compute  $r = \text{HashToIntegerRange}( SSV || b, q, \text{Hash} )$ ;
- 5) Compute  $\text{TEST} = [r]([b]P + Z_S)$  in  $E(F_p)$ . If  $\text{TEST}$  does not equal  $R_{(b,S)}$ , then  $B$  MUST NOT use the  $SSV$  to derive key material;
- 6) Output  $SSV$  for use to derive key material for the application to be keyed.

### 6.3. Group Communications

The SAKKE scheme can be used to exchange SSVs for group communications. To provide a shared secret to multiple Receivers, a Sender MUST form Encapsulated Data for each of their Identifiers and transmit the appropriate data to each Receiver. Any party possessing the group SSV MAY extend the group by forming Encapsulated Data for a new group member.

While the Sender needs to form multiple Encapsulated Data, the fact that the sending operation avoids pairings means that the extension to multiple Receivers can be carried out more efficiently than for alternative IBE schemes that require the Sender to compute a pairing.

## 7. Security Considerations

This document describes the SAKKE cryptographic algorithm. 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 SAKKE in the Random Oracle Model [SK-KEM].

When defining public parameters, guidance on parameter sizes from [SP800-57] SHOULD be followed. Note that the size of the  $F_p^2$  discrete logarithm on which the security rests is  $2 \cdot \lg(p)$ . Table 1 shows bits of security afforded by various sizes of  $p$ . If  $k$  bits of security are needed, then  $\lg(q)$  SHOULD be chosen to be at least  $2 \cdot k$ . Similarly, if  $k$  bits of security are needed, then a hash with output size at least  $2 \cdot k$  SHOULD be chosen.

Bits of Security		lg(p)
80		512
112		1024
128		1536
192		3840
256		7680

Table 1: Comparable Strengths, Taken from Table 2 of [SP800-57]

The KMS Master Secret provides the security for each device provisioned by the KMS. It MUST NOT be revealed to any other entity. Each user's RSK protects the SAKKE communications it receives. This key MUST NOT be revealed to any entity other than the trusted KMS and the authorized user.

In order to ensure that the RSK is received only by an authorized device, it MUST be provided through a secure channel. The security offered by this system is no greater than the security provided by this delivery channel.

Note that IBE systems have different properties than other asymmetric cryptographic schemes with regard to key recovery. The KMS (and hence any administrator with appropriate privileges) can create RSKs for arbitrary Identifiers, and procedures to monitor the creation of RSKs, such as logging of administrator actions, SHOULD be defined by any functioning implementation of SAKKE.

Identifiers MUST be defined unambiguously by each application of SAKKE. Note that it is not necessary to hash the data in a format for Identifiers (except in the case where its size would be greater than that of q). In this way, any weaknesses that might be caused by collisions in hash functions can be avoided without reliance on the structure of the Identifier format. Applications of SAKKE MAY include a time/date component in their Identifier format to ensure that Identifiers (and hence RSKs) are only valid for a fixed period of time.

The randomness of values stipulated to be selected at random in SAKKE, as described in this document, is essential to the security provided by SAKKE. If the ephemeral value r selected by the Sender is not chosen at random, then the SSV, which is used to provide key material for further communications, could be predictable. Guidance on the generation of random values for security can be found in [RFC4086].

## 8. References

### 8.1. Normative References

- [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.
- [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.
- [SK-KEM] Barbosa, M., Chen, L., Cheng, Z., Chimley, M., Dent, A., Farshim, P., Harrison, K., Malone-Lee, J., Smart, N., and F. Vercauteren, "SK-KEM: An Identity-Based KEM", submission for IEEE P1363.3, June 2006, (<http://grouper.ieee.org/groups/1363/IBC/submissions/Barbosa-SK-KEM-2006-06.pdf>).
- [SP800-57] Barker, E., Barker, W., Burr, W., Polk, W., and M. Smid, "Recommendation for Key Management - Part 1: General (Revised)", NIST Special Publication 800-57, March 2007.

### 8.2. Informative References

- [Barreto] Barreto, P., Kim, H., Lynn, B., and M. Scott, "Efficient Algorithms for Pairing-Based Cryptosystems", Advances in Cryptology - Crypto 2002, LNCS 2442, Springer-Verlag (2002), pp. 354-369.
- [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.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

- [RFC5091] Boyen, X. and L. Martin, "Identity-Based Cryptography Standard (IBCS) #1: Supersingular Curve Implementations of the BF and BB1 Cryptosystems", RFC 5091, December 2007.
- [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 for SAKKE with the public parameters defined in Appendix A of [RFC6509]. 'b' represents the Identifier of the Responder. The value "mask" is the value used to mask the SSV and is defined to be  
HashToIntegerRange( g<sup>r</sup>, 2<sup>n</sup>, Hash ).

```
// -----  
// The KMS generates:  
  
z      = AFF429D3 5F84B110 D094803B 3595A6E2 998BC99F  
  
Zx     = 5958EF1B 1679BF09 9B3A030D F255AA6A  
        23C1D8F1 43D4D23F 753E69BD 27A832F3  
        8CB4AD53 DDEF4260 B0FE8BB4 5C4C1FF5  
        10EFFE30 0367A37B 61F701D9 14AEF097  
        24825FA0 707D61A6 DFF4FBD7 273566CD  
        DE352A0B 04B7C16A 78309BE6 40697DE7  
        47613A5F C195E8B9 F328852A 579DB8F9  
        9B1D0034 479EA9C5 595F47C4 B2F54FF2  
  
Zy     = 1508D375 14DCF7A8 E143A605 8C09A6BF  
        2C9858CA 37C25806 5AE6BF75 32BC8B5B  
        63383866 E0753C5A C0E72709 F8445F2E  
        6178E065 857E0EDA 10F68206 B63505ED  
        87E534FB 2831FF95 7FB7DC61 9DAE6130  
        1EEACC2F DA3680EA 4999258A 833CEA8F  
        C67C6D19 487FB449 059F26CC 8AAB655A  
        B58B7CC7 96E24E9A 39409575 4F5F8BAE  
  
// -----  
// Creating Encapsulated Data  
  
b      = 3230 31312D30 32007465 6C3A2B34  
        34373730 30393030 31323300  
  
SSV    = 12345678 9ABCDEF0 12345678 9ABCDEF0
```

```

r      = HashToIntegerRange(
      12345678 9ABCDEF0 12345678 9ABCDEF0
      32303131 2D303200 74656C3A 2B343437
      37303039 30303132 3300, q, SHA-256 )

      = 13EE3E1B 8DAC5DB1 68B1CEB3 2F0566A4
      C273693F 78BAFFA2 A2EE6A68 6E6BD90F
      8206CCAB 84E7F42E D39BD4FB 131012EC
      CA2ECD21 19414560 C17CAB46 B956A80F
      58A3302E B3E2C9A2 28FBA7ED 34D8ACA2
      392DA1FF B0B17B23 20AE09AA EDFD0235
      F6FE0EB6 5337A63F 9CC97728 B8E5AD04
      60FADE14 4369AA5B 21662132 47712096

Rbx    = 44E8AD44 AB8592A6 A5A3DDCA 5CF896C7
      18043606 A01D650D EF37A01F 37C228C3
      32FC3173 54E2C274 D4DAF8AD 001054C7
      6CE57971 C6F4486D 57230432 61C506EB
      F5BE438F 53DE04F0 67C776E0 DD3B71A6
      29013328 3725A532 F21AF145 126DC1D7
      77ECC27B E50835BD 28098B8A 73D9F801
      D893793A 41FF5C49 B87E79F2 BE4D56CE

Rby    = 557E134A D85BB1D4 B9CE4F8B E4B08A12
      BABF55B1 D6F1D7A6 38019EA2 8E15AB1C
      9F76375F DD1210D4 F4351B9A 009486B7
      F3ED46C9 65DED2D8 0DADE4F3 8C6721D5
      2C3AD103 A10EBD29 59248B4E F006836B
      F097448E 6107C9ED EE9FB704 823DF199
      F832C905 AE45F8A2 47A072D8 EF729EAB
      C5E27574 B07739B3 4BE74A53 2F747B86

g^r    = 7D2A8438 E6291C64 9B6579EB 3B79EAE9
      48B1DE9E 5F7D1F40 70A08F8D B6B3C515
      6F2201AF FBB5CB9D 82AA3EC0 D0398B89
      ABC78A13 A760C0BF 3F77E63D 0DF3F1A3
      41A41B88 11DF197F D6CD0F00 3125606F
      4F109F40 0F7292A1 0D255E3C 0EBCCB42
      53FB182C 68F09CF6 CD9C4A53 DA6C74AD
      007AF36B 8BCA979D 5895E282 F483FCD6

```

```

mask = HashToIntegerRange(
    7D2A8438 E6291C64 9B6579EB 3B79EAE9
    48B1DE9E 5F7D1F40 70A08F8D B6B3C515
    6F2201AF FBB5CB9D 82AA3EC0 D0398B89
    ABC78A13 A760C0BF 3F77E63D 0DF3F1A3
    41A41B88 11DF197F D6CD0F00 3125606F
    4F109F40 0F7292A1 0D255E3C 0EBCCB42
    53FB182C 68F09CF6 CD9C4A53 DA6C74AD
    007AF36B 8BCA979D 5895E282 F483FCD6, 2^128, SHA-256 )

    = 9BD4EA1E 801D37E6 2AD2FAB0 D4F5BBF7

H = 89E0BC66 1AA1E916 38E6ACC8 4E496507

// -----
// Receiver processing

// Device receives Kb from the KMS

Kbx = 93AF67E5 007BA6E6 A80DA793 DA300FA4
    B52D0A74 E25E6E7B 2B3D6EE9 D18A9B5C
    5023597B D82D8062 D3401956 3BA1D25C
    0DC56B7B 979D74AA 50F29FBF 11CC2C93
    F5DFCA61 5E609279 F6175CEA DB00B58C
    6BEE1E7A 2A47C4F0 C456F052 59A6FA94
    A634A40D AE1DF593 D4FECF68 8D5FC678
    BE7EFC6D F3D68353 25B83B2C 6E69036B

Kby = 155F0A27 241094B0 4BFB0BDF AC6C670A
    65C325D3 9A069F03 659D44CA 27D3BE8D
    F311172B 55416018 1CBE94A2 A783320C
    ED590BC4 2644702C F371271E 496BF20F
    588B78A1 BC01ECBB 6559934B DD2FB65D
    2884318A 33D1A42A DF5E33CC 5800280B
    28356497 F87135BA B9612A17 26042440
    9AC15FEE 996B744C 33215123 5DECB0F5

// Device processes Encapsulated Data

w = 7D2A8438 E6291C64 9B6579EB 3B79EAE9
    48B1DE9E 5F7D1F40 70A08F8D B6B3C515
    6F2201AF FBB5CB9D 82AA3EC0 D0398B89
    ABC78A13 A760C0BF 3F77E63D 0DF3F1A3
    41A41B88 11DF197F D6CD0F00 3125606F
    4F109F40 0F7292A1 0D255E3C 0EBCCB42
    53FB182C 68F09CF6 CD9C4A53 DA6C74AD
    007AF36B 8BCA979D 5895E282 F483FCD6

```

```

SSV      = 12345678 9ABCDEF0 12345678 9ABCDEF0

r        = 13EE3E1B 8DAC5DB1 68B1CEB3 2F0566A4
          C273693F 78BAFFA2 A2EE6A68 6E6BD90F
          8206CCAB 84E7F42E D39BD4FB 131012EC
          CA2ECD21 19414560 C17CAB46 B956A80F
          58A3302E B3E2C9A2 28FBA7ED 34D8ACA2
          392DA1FF B0B17B23 20AE09AA EDFD0235
          F6FE0EB6 5337A63F 9CC97728 B8E5AD04
          60FADE14 4369AA5B 21662132 47712096

TESTx    = 44E8AD44 AB8592A6 A5A3DDCA 5CF896C7
          18043606 A01D650D EF37A01F 37C228C3
          32FC3173 54E2C274 D4DAF8AD 001054C7
          6CE57971 C6F4486D 57230432 61C506EB
          F5BE438F 53DE04F0 67C776E0 DD3B71A6
          29013328 3725A532 F21AF145 126DC1D7
          77ECC27B E50835BD 28098B8A 73D9F801
          D893793A 41FF5C49 B87E79F2 BE4D56CE

TESTy    = 557E134A D85BB1D4 B9CE4F8B E4B08A12
          BABF55B1 D6F1D7A6 38019EA2 8E15AB1C
          9F76375F DD1210D4 F4351B9A 009486B7
          F3ED46C9 65DED2D8 0DADE4F3 8C6721D5
          2C3AD103 A10EBD29 59248B4E F006836B
          F097448E 6107C9ED EE9FB704 823DF199
          F832C905 AE45F8A2 47A072D8 EF729EAB
          C5E27574 B07739B3 4BE74A53 2F747B86

```

```
TEST == Rb
```

```
// -----
// HashToIntegerRange( M, q, SHA-256 ) example
```

```

M        = 12345678 9ABCDEF0 12345678 9ABCDEF0
          32303131 2D303200 74656C3A 2B343437
          37303039 30303132 3300

A        = E04D4EF6 9DF86893 22B39AE3 80284617
          4A93BEDB 1E3D2A2C 5F2C7EA0 05513EBA

h0       = 00000000 00000000 00000000 00000000
          00000000 00000000 00000000 00000000

h1       = 66687AAD F862BD77 6C8FC18B 8E9F8E20
          08971485 6EE233B3 902A591D 0D5F2925

```

```

h2      = 2B32DB6C 2C0A6235 FB1397E8 225EA85E
          0F0E6E8C 7B126D00 16CCBDE0 E667151E

h3      = 12771355 E46CD47C 71ED1721 FD5319B3
          83CCA3A1 F9FCE3AA 1C8CD3BD 37AF20D7

h4      = FE15C0D3 EBE314FA D720A08B 839A004C
          2E6386F5 AECC19EC 74807D19 20CB6AEB

v1      = FA2656CA 1D2DBD79 015AE918 773DFEDC
          24957C91 E3C9C335 40D6BF6D 7C3C0055

v2      = F016CD67 59620AD7 87669E3A DD887DF6
          25895A91 0CEE1486 91A06735 B2F0A248

v3      = AC45C6F9 7F83BCE0 A2BBD0A1 4CF4D7F4
          CB3590FB FAF93AE7 1C64E426 185710B5

v4      = E65D50BD 551A54EF 981F535E 072DE98D
          2223ACAD 4621E026 3B0A61EA C56DB078

v mod q = 13EE3E1B 8DAC5DB1 68B1CEB3 2F0566A4
          C273693F 78BAFFA2 A2EE6A68 6E6BD90F
          8206CCAB 84E7F42E D39BD4FB 131012EC
          CA2ECD21 19414560 C17CAB46 B956A80F
          58A3302E B3E2C9A2 28FBA7ED 34D8ACA2
          392DA1FF B0B17B23 20AE09AA EDFD0235
          F6FE0EB6 5337A63F 9CC97728 B8E5AD04
          60FADE14 4369AA5B 21662132 47712096

```

// -----

Author's Address

Michael Groves  
 CESG  
 Hubble Road  
 Cheltenham  
 GL51 8HJ  
 UK

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

