ESIGN-TSH Specification
Draft 1.0

NTT Information Sharing Platform Laboratories,
NTT Corporation

May 24, 2002

Contents

1 Introduction

This document provides a specification for implementing ESIGN-TSH, which is a signature scheme with appendix.

This specification is compatible with the IFSSA scheme in the IEEE P1363a / D10 [1], where the signature and verification primitives are IFSP-ESIGN and IFVP-ESIGN, respectively, and the message encoding method is EMSA5.

In this document, SSA-ESIGN-TSH-Sign and SSA-ESIGN-TSH-Verify describe the algorithms of ESIGN-TSH. They consist of

- cryptographic primitives
  - Key generation primitive: KGP-ESIGN-TSH,
  - Signtature primitive: SP-ESIGN-TSH,
  - Verification primitive: VP-ESIGN-TSH, and
- an encoding method
  - EMSA-ESIGN-TSH.

2 Notation

bit	one of the two symbols 0 or 1.
bit string	an ordered sequence of bits.
octet	one of the integers from 0 to 255.
octet string	an ordered sequence of octets.
$\boldsymbol{R}$	the set of real numbers.
$\boldsymbol{Z}$	the set of integers.
$\boldsymbol{N}$	the set of positive integers.
$a := b$	assign b to a.
$(B_0, B_1, \ldots, B_{i-1})_2$	a bit string of length i, for example, $(0, 1, 0, 0)_2$.
$(M_0, M_1, \ldots, M_{i-1})_{256}$	an octet string of length i, for example, $(170, 255, 0)_{256}$.
$\{0,1\}^i$	if $i \in \boldsymbol{N}$, then the set of all bit strings of length i. if $i = 0$, then empty bit string.
$\{0,1\}^*$	$\bigcup_{i=0}^{\infty} \{0,1\}^i$.
$\{0, 1, \ldots, 255\}^i$	if $i \in \boldsymbol{N}$, then the set of all octet strings of length i. if $i = 0$, then empty octet string.
$\{0, 1, \ldots, 255\}^*$	$\bigcup_{i=0}^{\infty} \{0, 1, \ldots, 255\}^i$.
$\|$	a concatenation operator for two bit strings or for two octet strings, for example, $(0, 1, 0, 0)_2 \,\|\, (1, 1, 0)_2 = (0, 1, 0, 0, 1, 1, 0)_2$ for bit strings, $(170, 255)_{256} \,\|\, (0, 20)_{256} = (170, 255, 0, 20)_{256}$ for octet strings. The operator is often omitted.
$\lceil y \rceil$	for $y \in \boldsymbol{R}$, the least integer greater than or equal to y.
$\lfloor y \rfloor$	for $y \in \boldsymbol{R}$, the greatest integer less than or equal to y.
$[X]_y$	for $y \in \boldsymbol{N}$, the rightmost y bits of a bit string X.
$\mathrm{GCD}(a, b)$	for $a \in \boldsymbol{N}$, $b \in \boldsymbol{N}$, the greatest common divisor of a and b.
$a \mid b$	for $a \in \boldsymbol{Z}$, $b \in \boldsymbol{Z}$, a divides b.
$a \bmod m$	for $a \in \boldsymbol{Z}$, $m \in \boldsymbol{N}$, the least nonnegative integer b which satisfies $m \mid (b - a)$.
$a^{-1} \bmod m$	for $a \in \boldsymbol{Z}$, $m \in \boldsymbol{N}$, the least nonnegative integer b which satisfies $ab \bmod m = 1$.

3 Data types and conversions

The schemes specified in this document involve operations using several different data types. Figure 1 illustrates which conversions are needed and where they are described.

3.1 Integer-to-BitString Conversion(I2BSP)

Integers should be converted to bit strings as described in this section. Informally, the idea is to represent the integer in binary. Formally, the conversion routine, $\mathrm{I2BSP}(x, l)$, is specified as follows:

Input:	x	an integer to be converted, a nonnegative integer
	l	the bit length of the output, a nonnegative integer
Output:	B	a bit string of length l bits
Errors:	`INVALID`	
Steps:		

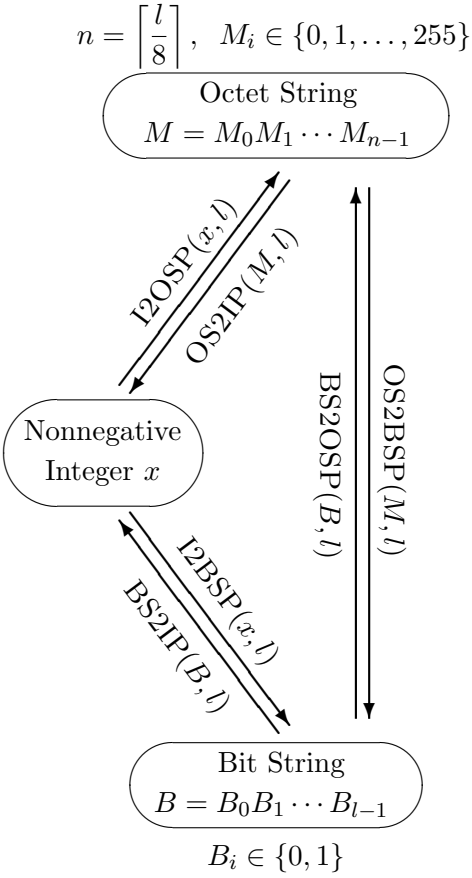

Figure 1: Conversion between data types

1. If $l = 0$, output an empty bit string and stop.
2. If $x \geq 2^l$, assert `INVALID` and stop.
3. Determine the x's base-2 representation, $x_i \in \{0, 1\}$ such that
$$x = x_{l-1}2^{l-1} + x_{l-2}2^{l-2} + \cdots + x_1 2 + x_0.$$
4. For $0 \leq i \leq l - 1$, set $B_i := x_{l-1-i}$, and let
$$B := B_0 B_1 \cdots B_{l-1}.$$
5. Output B.

3.2 BitString-to-Integer Conversion(BS2IP)

Bit strings should be converted to integers as described in this section. Informally, the idea is simply to view the bit string as the base-2 representation of the integer. Formally, the conversion routine, BS2IP(B, l), is specified as follows:

Input:	B	a bit string to be converted
	l	the bit length of B, a nonnegative integer
Output:	x	a nonnegative integer
Steps:		

Convert $B = B_0B_1 \cdots B_{l-1}$ to an integer x as follows:

1. If $l = 0$, output 0 and stop.
2. View each B_i as an integer in $\{0, 1\}$, set $x_i := B_i$ for $0 \leq i \leq l - 1$, and let
$$x := \sum_{i=0}^{l-1} 2^{(l-1-i)} x_i.$$
3. Output x.

3.3 BitString-to-OctetString Conversion(BS2OSP)

Bit strings should be converted to octet strings as described in this section. Informally, the idea is to pad the bit string with 0's on the left to make its length a multiple of 8, then chop the result up into octets. Formally, the conversion routine, BS2OSP(B, l), is specified as follows:

Input: B a bit string to be converted
l the bit length of B, a nonnegative integer
Output: M an octet string of length $n = \left\lceil \frac{l}{8} \right\rceil$ octets
Steps:

Convert the bit string $B = B_0B_1 \cdots B_{l-1}$ to an octet string $M = M_0M_1 \cdots M_{n-1}$ as follows:

1. If $l = 0$, output an empty octet string and stop.
2. For $0 < i \leq n - 1$, let:
$$M_i := B_{l-8-8(n-1-i)}B_{l-7-8(n-1-i)} \cdots B_{l-1-8(n-1-i)}.$$
3. Set
$$M_0 := \begin{cases} B_0B_1 \cdots B_7 & (8n - l = 0) \\ ZB_0B_1 \cdots B_{l+7-8n} & (8n - l \neq 0) \end{cases},$$
where Z is an all zero bit string of length $8n - l$ ($Z = (0, 0, \ldots, 0)_2$).
4. Output M.

3.4 OctetString-to-BitString Conversion(OS2BSP)

Octet strings should be converted to bit strings as described in this section. Informally, the idea is simply to view the octet string as a bit string. Formally, the conversion routine, OS2BSP(M, l), is specified as follows:

Input: M an octet string of length $n = \left\lceil \frac{l}{8} \right\rceil$ octets to be converted
l the bit length of the output, a nonnegative integer
Output: B a bit string of length l bits
Steps:

Convert the octet string $M = M_0M_1 \cdots M_{n-1}$ to a bit string $B = B_0B_1 \cdots B_{l-1}$ as follows:

1. If $l = 0$, output an empty bit string and stop.
2. For $0 < i \leq n - 1$, set:
$$B_{l-8-8(n-1-i)}B_{l-7-8(n-1-i)} \cdots B_{l-1-8(n-1-i)} := M_i.$$
3. For $0 \leq j \leq l + 7 - 8n$, set $B_j := Z_{j+8n-l}$, where $Z_0Z_1 \cdots Z_7 := M_0$.
4. Output B.

3.5 Integer-to-OctetString Conversion(I2OSP)

Integers should be converted to octet strings as described in this section. Informally, the idea is to represent the integer in binary and then convert the resulting bit string to an octet string. Formally, the conversion routine, I2OSP(x, l), is specified as follows:

Input: x an integer to be converted, a nonnegative integer
l the bit length of x, a nonnegative integer
Output: M an octet string of length $n = \left\lceil \frac{l}{8} \right\rceil$ octets
Errors: `INVALID`
Steps:

1. If $l = 0$, output an empty octet string and stop.
2. If $x \geq 2^l$, assert `INVALID` and stop.
3. Determine the x's base-256 representation, $x_i \in \{0, \ldots, 255\}$ such that
$$x = x_{n-1}2^{8(n-1)} + x_{n-2}2^{8(n-2)} + \cdots + x_1 2^8 + x_0.$$
4. For $0 \leq i \leq n - 1$, set $M_i := x_{n-1-i}$, and let
$$M := M_0M_1 \cdots M_{n-1}.$$
5. Output M.

3.6 OctetString-to-Integer Conversion(OS2IP)

Octet strings should be converted to integers as described in this section. Informally, the idea is simply to view the octet string as the base-256 representation of the integer. Formally, the conversion routine, OS2IP(M, l), is specified as follows:

Input: M an octet string of length $n = \left\lceil \frac{l}{8} \right\rceil$ octets to be converted
l the bit length of x, a nonnegative integer
Output: x a nonnegative integer
Steps:

Convert $M = M_0M_1 \cdots M_{n-1}$ to an integer, x, as follows:

1. If $l = 0$, output 0 and stop.
2. View each M_i as an integer in $\{0, \ldots, 255\}$, set $x_i := M_i$ for $0 \leq i \leq l - 1$, and let
$$x := \sum_{i=0}^{n-1} 2^{8(n-1-i)}x_i \mod 2^l.$$
3. Output x.

4 Key types

In this section, two key types are defined: ESIGN private key and ESIGN public key.

4.1 ESIGN private key

An ESIGN private key is the 5-tuple $(p, q, n, pLen, e)$, where the components have the following meanings:

- p, , the first factor, a prime number
- q, the second factor, a prime number
- n, the modulus, a positive integer
- $pLen$, the security parameter, a positive integer
- e, the exponent, a positive integer

Valid ESIGN private key $2^{pLen-1} < p < 2^{pLen}$, $2^{pLen-1} < q < 2^{pLen}$, $n = p^2q$, and $2^{3pLen-1} < n < 2^{3pLen}$.

4.2 ESIGN public key

An ESIGN public key is the 3-tuple $(n, pLen, e)$, where the components have the following meanings:

- n, the modulus, a positive integer
- $pLen$, the security parameter, a positive integer
- e, the exponent, a positive integer

Valid ESIGN public key $2^{3pLen-1} < n < 2^{3pLen}$.

5 Cryptographic primitives

In this section, three cryptographic primitives are specified.

5.1 KGP-ESIGN-TSH

KGP-ESIGN-TSH(k, e) is defined as follows:

Input:	k	security parameter, a positive integer
	e	public exponent, a positive integer
Output:	PK	ESIGN public key $(n, pLen, e)$
	SK	ESIGN private key $(p, q, n, pLen, e)$
Errors:	`INVALID`	

Steps:

1. If $e < 8$, assert `INVALID` and stop.
2. Choose two odd primes p, q such that $2^{k-1} < p < 2^k$, $2^{k-1} < q < 2^k$, $p \neq q$, $2^{3k-1} < p^2q < 2^{3k}$, and compute $n := p^2q$.

3. Set $pLen := k$.
4. Output $PK = (n, pLen, e)$, $SK = (p, q, n, pLen, e)$.

5.2 SP-ESIGN-TSH

SP-ESIGN-TSH(SK, f) is defined as follows:

Input:	SK	ESIGN private key $(p, q, n, pLen, e)$
	f	message representative, an integer, $0 \leq f < 2^{pLen-1}$
Output:	s	signature representative, an integer, $0 \leq s < n$
Errors:	`INVALID`	
Assumptions:	private key SK is valid	
Steps:		

1. If the message representative f does not satisfy $0 \leq f < 2^{pLen-1}$, assert `INVALID` and stop.
2. Compute $z := f \cdot 2^{2pLen}$.
3. Choose a random integer $r \in \{1, 2, \ldots, pq - 1\}$ such that $\mathrm{GCD}(r, n) = 1$.
4. Compute $\alpha := (z - r^e) \bmod n$.
5. Let (w_0, w_1) be
$$\begin{aligned} w_0 &:= \left\lceil \frac{\alpha}{pq} \right\rceil, \\ w_1 &:= w_0 pq - \alpha. \end{aligned}$$
6. If $w_1 \geq 2^{2pLen-1}$, go back to step 3.
7. Let $t := w_0 (e r^{e-1})^{-1} \bmod p$, and compute $s := r + tpq$.
8. Output s.

Note: If the assumptions do not hold, the performance of this function is not guaranteed.

5.3 VP-ESIGN-TSH

VP-ESIGN-TSH(PK, s) is defined as follows:

Input:	PK	ESIGN public key $(n, pLen, e)$
	s	signature representative, an integer, $0 \leq s < n$
Output:	f	message representative, an integer, $0 \leq f < 2^{pLen-1}$
Errors:	`INVALID`	
Assumptions:	public key PK is valid	
Steps:		

1. If the signature representative s does not satisfy $0 \leq s < n$, assert `INVALID` and stop.
2. Compute $T := s^e \bmod n$.
3. Compute $f := \left\lfloor \frac{T}{2^{2pLen}} \right\rfloor$.
4. If f does not satisfy $0 \leq f < 2^{pLen-1}$, assert `INVALID` and stop.
5. Output f.

Note: If the assumptions do not hold, the performance of this function is not guaranteed.

6 Signature schemes with appendix

In this section, one signature scheme with appendix is specified.

6.1 SSA-ESIGN-TSH

SSA-ESIGN-TSH combines the SP-ESIGN-TSH and VP-ESIGN-TSH primitives with the EMSA-ESIGN-TSH encoding method.

6.1.1 Signature generation operation

SSA-ESIGN-TSH-SIGN(SK, M) is defined as follows:

Input:	SK	signer's ESIGN private key
	M	message to be signed, an octet string
Output:	s	signature representative, an integer, $0 \leq s < n$
Errors:	`INVALID`	
Steps:		

1. Apply the EMSA-ESIGN-TSH-ENCODE operation (Section 7.1.1) to the message M to produce an integer message representative f:

   $$f := \text{EMSA-ESIGN-TSH-ENCODE}(M, pLen - 1).$$

   If the encoding operation asserts `INVALID`, then assert `INVALID` and stop.

2. Apply the SP-ESIGN-TSH signature primitive (Section 5.2) to the private key SK and the message representative f to produce an integer signature representative s:

   $$s := \text{SP-ESIGN-TSH}(SK, f).$$

3. Output the signature representative s.

6.1.2 Signature verification operation

SSA-ESIGN-TSH-VERIFY(PK, M, s) is defined as follows:

Input:	PK	signer's ESIGN public key
	M	message whose signature is to be verified, an octet string
	s	signature to be verified, an integer, $0 \leq s < n$
Output:	`VALID_SIGNATURE` or `INVALID_SIGNATURE`	
Steps:		

1. Apply the VP-ESIGN-TSH verification primitive (Section 5.3) to the public key PK and the signature representative s to produce an integer message representative f':

   $$f' := \text{VP-ESIGN-TSH}(PK, s).$$

   If the VP-ESIGN-TSH primitive asserts `INVALID`, then output `INVALID_SIGNATURE` and stop.

2. Apply the EMSA-ESIGN-TSH-VERIFY operation (Section 7.1.2) to the message M and the message representative f' to determine whether they are consistent:

$$Result := \text{EMSA-ESIGN-TSH-VERIFY}(M, f', pLen - 1).$$

If *Result* is `CONSISTENT`, then output `VALID_SIGNATURE`.
Otherwise, output `INVALID_SIGNATURE`.

7 Encoding methods

In this section, one encoding method is specified.

7.1 EMSA-ESIGN-TSH

This encoding method is compatible with the EMSA5 encoding method in the IEEE P1363a / D10 [1].

7.1.1 Encoding operation

EMSA-ESIGN-TSH-ENCODE(M, l) is defined as follows:

System parameters:	*Hash*	hash function
	hLen	length in bits of the hash function output, a positive integer
	MGF	mask generation function
Input:	M	message to be encoded, an octet string
	l	the bit length of the encoded message, a positive integer
Output:	f	encoded message representative, a nonnegative integer
Errors:	`INVALID`	
Steps:		

1. If the length of M is greater than the input limitation for the hash function, assert `INVALID` and stop.

2. Apply the hash function to the message M to produce a hash value H of length $\left\lceil \frac{hLen}{8} \right\rceil$ octets:

$$H := Hash(M).$$

3. Apply the mask generation function to H to produce an octet string T of length $\left\lceil \frac{l}{8} \right\rceil$ octets:

$$T := MGF(H, l).$$

4. Convert the octet string T to an integer f:

$$f := \text{OS2IP}(T, l).$$

5. Output f.

7.1.2 Verification operation

EMSA-ESIGN-TSH-VERIFY(M, f, l) is defined as follows:

System parameters:	*Hash*	hash function
	hLen	length in bits of the hash function output, a positive integer
	MGF	mask generation function
Input:	M	message to be encoded, an octet string
	f	encoded message representative, a nonnegative integer
	l	the bit length of f, a positive integer
Output:	`CONSISTENT`	
Errors:	`INVALID`	
Steps:		

1. Convert the encoded message representative f to a bit string T:

$$T := \text{I2BSP}(f, l).$$

   If the I2BSP primitive asserts `INVALID`, then assert `INVALID` and stop.

2. If the length of M is greater than the input limitation for the hash function, assert `INVALID` and stop.

3. Apply the hash function to M to produce a hash value H of length $\left\lceil \frac{hLen}{8} \right\rceil$ octets:

$$H := Hash(M).$$

4. Apply the mask generation function to H to produce an octet string T' of length $\left\lceil \frac{l}{8} \right\rceil$ octets:

$$T' := MGF(H, l).$$

5. Convert the octet string T' to a bit string T'':

$$T'' := \text{OS2BSP}(T', l).$$

6. If T is equal to T'', then output `CONSISTENT`. Otherwise assert `INVALID`.

8 Auxiliary techniques

This section gives several examples of the techniques that support the functions described in this document.

8.1 Hash functions

One hash function is recommended for the encoding methods in this document: SHA-1.

8.1.1 SHA-1

SHA-1 is defined in FIPS PUB 180-1 [2]. The output length of SHA-1 is 160 bits, and the operation block size is 512 bits.

8.2 Mask generation functions

One mask generation function is recommended for the encoding methods in this document: MGF1 [3].

8.2.1 MGF1

MGF1 is a mask generation function based on a hash function.

MGF1(M, l) is defined as follows:

System parameters:	*Hash*	hash function
	hashLen	length in bits of the hash function output, a positive integer
Input:	M	seed from which mask is generated, an octet string
	l	the bit length of the output, a positive integer
Output:	*mask*	mask, an octet string of length $\left\lceil \frac{l}{8} \right\rceil$ octets
Errors:	`INVALID`	
Steps:		

1. Let l_0 be the bit length of M. If $l_0 + 32$ is greater than the input limitation for the hash function, assert `INVALID` and stop.
2. Let $cThreshold := \left\lceil \frac{l}{hashLen} \right\rceil$.
3. Let M' be the empty octet string.
4. Let $counter := 0$.
   (a) Convert the integer *counter* to an octet string C of length 32 bits:
   $$C := \text{I2OSP}(counter, 32).$$
   (b) Concatenate M and C, and apply the hash function to the result to produce a hash value H of length $\left\lceil \frac{hashLen}{8} \right\rceil$ octets:
   $$H := Hash(M \,\|\, C).$$
   (c) Concatenate M' and H to the octet string M':
   $$M' := M' \,\|\, H.$$
   (d) Let $counter := counter + 1$. If $counter < cThreshold$, go back to step 4a.
5. Let *mask* be the leftmost $\left\lceil \frac{l}{8} \right\rceil$ octets of the octet string M':
   $$mask := M'_0 M'_1 \cdots M'_{\lceil l/8 \rceil - 1}.$$
6. Output *mask*.

References

[1] IEEE P1363a / D10 (Draft Version 10), "Standard Specifications for Public Key Cryptography: Additional Techniques," , IEEE, to be appeared.

[2] FIPS PUB 180-1, "Secure Hash Standard (SHS)," U.S. Department of Commerce / National Institute of Standards and Technology, April 17, 1995.

[3] RSA Laboratories, "PKCS #1 v2.1: RSA Encryption Standard," draft 2, January 5, 2001.

Appendix

A Security requirements of parameters

Security requirements of ESIGN-TSH parameters are the following:

$$\begin{aligned} k &\geq 342 \qquad \text{(bit length of } n \geq 1024\text{)}. \\ e &\geq 8 \end{aligned}$$

B Recommended values of parameters

Recommended values of ESIGN-TSH parameters are the following:

$$\begin{aligned} k &= 384 \qquad \text{(bit length of } n = 1152\text{)} \\ e &= 1024 \\ \mathit{Hash} &= \text{SHA-1} \\ \mathit{hLen} &= 160 \\ \mathit{MGF} &= \text{MGF1} \qquad \text{(SHA-1, } \mathit{hashLen} = 160\text{)} \end{aligned}$$

C Recommended usage of message M

We recommend each message, M, to be signed only once, since our formal proof guarantees the security against OM-CMA instead of CMA (see Self-Evaluation of ESIGN Theorem 2.3)

So, a plain massage M_0 to be signed should be padded by one-time data, OT, such as a sequential number counter for each signer, a time stamp, a random number.

That is,

$$M = M_0 \,\|\, OT, \qquad OT \text{ is an one-time octet string.}$$

Examples of OT:

- OT is an octet string of length 20 octets:

$$\begin{aligned} OT &= \text{I2OSP}(c, 160). \\ c &: \text{a sequential number for each signer}, 0 \leq c < 2^{160}. \end{aligned}$$

- OT is an octet string of length 20 octets:

$$\begin{aligned} OT &= \text{I2OSP}(ps, 160). \\ ps &: \text{a time since the 00:00:00.000000000000 UTC, January 1, 1970,} \\ &\quad \text{measured in pico seconds}, 0 \leq ps < 2^{160}. \end{aligned}$$

- OT is an octet string of length 20 octets:

$$\begin{aligned} OT &= \text{I2OSP}(r, 160). \\ r &: \text{a random number}, 0 \leq r < 2^{160}. \end{aligned}$$