デジタル署名 ESIGN-TSH 仕様書
ドラフト 1.0

日本電信電話株式会社
NTT 情報流通プラットフォーム研究所

2002 年 5 月 23 日

目 次

1 はじめに

本ドキュメントは, デジタル署名方式の一つ, ESIGN-TSH 署名の実装に関する情報をまとめたものである.

本ドキュメントに記述されている仕様は, IEEE P1363a / D10 [1] に記述されている署名プリミティブ IFSP-ESIGN, 検証プリミティブ IFVP-ESIGN, エンコーディングメソッド EMSA5 とそれぞれ互換性がある.

本ドキュメントにおいて, ESIGN-TSH 署名アルゴリズムは, スキーム SSA-ESIGN-TSH-SIGN と SSA-ESIGN-TSH-VERIFY に記述されており, それらは以下から構成される.

- プリミティブ
  - 鍵生成プリミティブ: KGP-ESIGN-TSH
  - 署名プリミティブ: SP-ESIGN-TSH
  - 検証プリミティブ: VP-ESIGN-TSH
- エンコーディングメソッド
  - EMSA-ESIGN-TSH

2 記法

ビット	0 または 1 で表される記号.
ビット列	順序付けられたビットの系列.
オクテット	0 から 255 の整数.
オクテット列	順序付けられたオクテットの系列.
$\boldsymbol{R}$	実数の集合.
$\boldsymbol{Z}$	整数の集合.
$\boldsymbol{N}$	正の整数の集合.
$a := b$	b の値を a に代入する. または, a は b で定義される.
$(B_0, B_1, \ldots, B_{i-1})_2$	$i \in \boldsymbol{N}$ のとき, 長さ i のビット列. 例えば, $(0,1,0,0)_2$.
$(M_0, M_1, \ldots, M_{i-1})_{256}$	$i \in \boldsymbol{N}$ のとき, 長さ i のオクテット列. 例えば, $(170, 255, 00)_{256}$.
$\{0,1\}^i$	$i \in \boldsymbol{N}$ のとき, 長さ i のビット列の集合. $i = 0$ のとき, 空ビット列.
$\{0,1\}^*$	$\bigcup_{i=0}^{\infty} \{0,1\}^i$.
$\{0,1,\ldots,255\}^i$	$i \in \boldsymbol{N}$ のとき, 長さ i のオクテット列の集合. $i = 0$ のとき, 空オクテット列.
$\{0,1,\ldots,255\}^*$	$\bigcup_{i=0}^{\infty} \{0,1,\cdots,255\}^i$.
$\|$	ビット列またはオクテット列の連結演算子. 例えば, ビット列に対して $(0,1,0,0)_2 \,\|\, (1,1,0)_2 = (0,1,0,0,1,1,0)_2$, オクテット列に対して $(170,255)_{256} \,\|\, (0,20)_{256} = (170,255,0,20)_{256}$ となる. この演算子は, しばしば省略される.
$\lceil y \rceil$	$y \in \boldsymbol{R}$ のとき, y 以上の最小の整数.
$\lfloor y \rfloor$	$y \in \boldsymbol{R}$ のとき, y 以下の最大の整数.
$[X]_y$	$y \in \boldsymbol{N}$ のとき, ビット列 X の下位 y ビットを表す.
$\mathrm{GCD}(a,b)$	$a \in \boldsymbol{N}$, $b \in \boldsymbol{N}$ のとき, a と b の最大公約数.
$a \mid b$	$a \in \boldsymbol{Z}$, $b \in \boldsymbol{Z}$ のとき, a が b を割り切る.
$a \bmod m$	$a \in \boldsymbol{Z}$, $m \in \boldsymbol{N}$ のとき, $m \mid (b-a)$ なる最小の非負整数 b.
$a^{-1} \bmod m$	$a \in \boldsymbol{Z}$, $m \in \boldsymbol{N}$ のとき, $ab \bmod m = 1$ となる最小の非負整数 b.

3 データ型 及び 変換

本ドキュメントで規定するスキームには, さまざまなデータ型の演算が使用される. 必要な変換と対応する変換の通称を 図 1 に記載する.

3.1 整数からビット列への変換 (I2BSP)

本章に記述する整数からビット列への変換方法は, 基本的には整数の 2 進表現に基づいている. プリミティブ $\mathrm{I2BSP}(x, l)$ の詳細は以下の通りである.

入力:

x : 変換される整数, 非負整数
l : 出力されるビット長, 非負整数

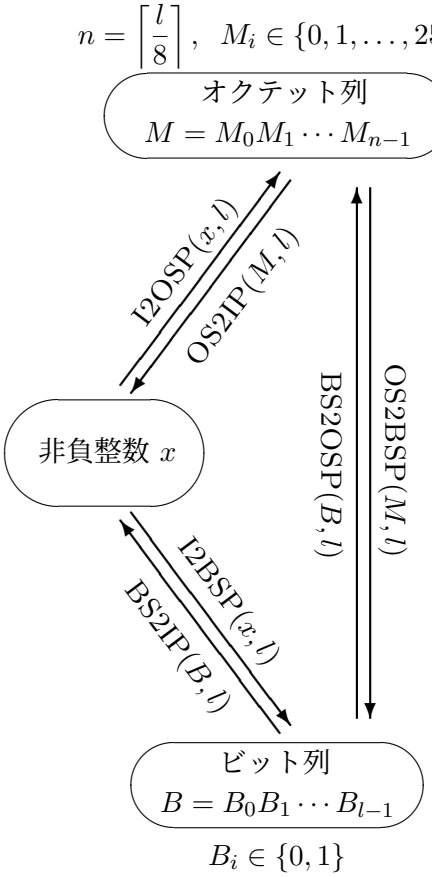

図 1: データ型間の変換

出力:

B : l ビット長のビット列

エラー:

`INVALID`

処理手順:

1. $l = 0$ ならば, 空ビット列を出力して処理を終了する.
2. $x \geq 2^l$ ならば, `INVALID` をエラー出力して処理を終了する.
3. $0 \leq i \leq l-1$ の整数 i に関して次を満たす x の 2 進表現 $x_i \in \{0, 1\}$ を求める:

$$x = x_{l-1} 2^{l-1} + x_{l-2} 2^{l-2} + \cdots + x_1 2 + x_0.$$

4. $0 \leq i \leq l-1$ の i について $B_i := x_{l-1-i}$ として以下とする:

$$B := B_0 B_1 \cdots B_{l-1}.$$

5. B を出力する.

3.2 ビット列から整数への変換 (BS2IP)

本章に記述するビット列から整数への変換方法は, 基本的にはビット列を単に整数の 2 進表現と見なす事に基づいている. プリミティブ BS2IP(B, l) の詳細は以下の通りである.

入力:

B : 変換されるビット列
l : B のビット長, 非負整数

出力:

x : 非負整数

処理手順:

ビット列 $B = B_0B_1 \cdots B_{l-1}$ を整数 x に以下のように変換する:

1. $l = 0$ ならば, 0 を出力して処理を終了する.
2. 各 B_i を 整数 0 または 1 と見なし, $0 \leq i \leq l-1$ の i について $x_i := B_i$ として以下とする:
$$x := \sum_{i=0}^{l-1} 2^{(l-1-i)} x_i.$$
3. x を出力する.

3.3 ビット列からオクテット列への変換 (BS2OSP)

本章に記述するビット列からオクテット列への変換方法は, 基本的にはビット列の長さが 8 の倍数となるまでビット 0 をビット列の先頭に付与し, しかる後にビット列を 8 ビットずつ切り出すという操作に基づいている. プリミティブ BS2OSP(B, l) の詳細は以下の通りである.

入力:

B : 変換されるビット列
l : B のビット長, 非負整数

出力:

M : $n = \left\lceil \frac{l}{8} \right\rceil$ オクテット長のオクテット列

処理手順:

ビット列 $B = B_0B_1 \cdots B_{l-1}$ をオクテット列 $M = M_0M_1 \cdots M_{n-1}$ へ以下の様に変換する:

1. $l = 0$ ならば, 空オクテット列を出力して処理を終了する.
2. $0 < i \leq n-1$ の 整数 i に関して以下とする:
$$M_i := B_{l-8-8(n-1-i)} B_{l-7-8(n-1-i)} \cdots B_{l-1-8(n-1-i)}.$$
3. M_0 を以下とする:
$$M_0 := \begin{cases} B_0B_1 \cdots B_7 & (8n - l = 0) \\ ZB_0B_1 \cdots B_{l+7-8n} & (8n - l \neq 0) \end{cases},$$
ここで, Z はビット長が $8n - l$ で, 全ビットが 0 であるビット列である. $(Z = (0, 0, \ldots, 0)_2)$.

4. M を出力する.

3.4 オクテット列からビット列への変換 (OS2BSP)

本章に記述するオクテット列からビット列への変換方法は, 基本的にはオクテット列を単に先頭にパディングビットを持つビット列と見なす事に基づいている. プリミティブ OS2BSP(M, l) の詳細は以下の通りである.

入力:

M : 変換される $n = \left\lceil \frac{l}{8} \right\rceil$ オクテット長のオクテット列
l : 出力されるビット長, 非負整数

出力:

B : l ビット長のビット列

処理手順:

オクテット列 $M = M_0 M_1 \cdots M_{n-1}$ をビット列 $B = B_0 B_1 \cdots B_{l-1}$ へ以下の様に変換する:

1. $l = 0$ ならば, 空ビット列を出力して処理を終了する.
2. $0 < i \leq n-1$ の整数 i に関して以下とする:

$$B_{l-8-8(n-1-i)} B_{l-7-8(n-1-i)} \cdots B_{l-1-8(n-1-i)} := M_i.$$

3. $0 \leq j \leq l+7-8n$ の整数 i に関して, $B_j := Z_{j+8n-l}$ とする. ここで, $Z_0 Z_1 \cdots Z_7 := M_0$ である.
4. B を出力する.

3.5 整数からオクテット列への変換 (I2OSP)

本章に記述する整数からオクテット列への変換方法は, 基本的には整数の 256 進表現に基づいている. プリミティブ I2OSP(x, l) の詳細は以下の通りである.

入力:

x : 変換される整数, 非負整数
l : x のビット長, 非負整数

出力:

M : $n = \left\lceil \frac{l}{8} \right\rceil$ オクテット長のオクテット列

エラー:

`INVALID`

処理手順:

1. $l = 0$ ならば, 空オクテット列を出力して処理を終了する.
2. $x \geq 2^l$ ならば, `INVALID` をエラー出力して処理を終了する.

3. $0 \leq i \leq n-1$ の整数 i に関して 次を満たす x の 256 進表現 $x_i \in \{0, \ldots, 255\}$ を求める:

$$x = x_{n-1}2^{8(n-1)} + x_{n-2}2^{8(n-2)} + \cdots + x_1 2^8 + x_0.$$

4. $0 \leq i \leq n-1$ の整数 i に関して $M_i := x_{n-1-i}$ として以下とする:

$$M := M_0 M_1 \cdots M_{n-1}.$$

5. M を出力する.

3.6 オクテット列から整数への変換 (OS2IP)

本章に記述するオクテット列から整数への変換方法は, 基本的にはオクテット列を単に整数の 256 進表現と見なす事に基づいている. プリミティブ OS2IP(M, l) の詳細は以下の通りである.

入力:

M : 変換される $n = \left\lceil \frac{l}{8} \right\rceil$ オクテット長のオクテット列
l : x のビット長, 非負整数

出力:

x : 非負整数

処理手順:

オクテット列 $M = M_0 M_1 \cdots M_{n-1}$ を 整数 x に以下のように変換する:

1. $l = 0$ ならば, 0 を出力して処理を終了する.
2. 各 M_i を 0 以上 255 以下の整数と見なし, $0 \leq i \leq l-1$ の i について $x_i := M_i$ として以下とする:

$$x := \sum_{i=0}^{n-1} 2^{8(n-1-i)} x_i \mod 2^l.$$

3. x を出力する.

4 鍵タイプ

本章では, ESIGN 秘密鍵および ESIGN 公開鍵の 2 種類の鍵タイプを定義する.

4.1 ESIGN 秘密鍵

ESIGN 秘密鍵は次の 5 つ組からなる.

- 素数 p
- 素数 q
- モジュラス n, 正の整数
- セキュリティパラメータ $pLen$, 正の整数
- 指数 e, 正の整数

正しい ESIGN 秘密鍵　$2^{pLen-1} < p < 2^{pLen},\ 2^{pLen-1} < q < 2^{pLen},\ n = p^2q,\ 2^{3pLen-1} < n < 2^{3pLen}$ が成り立つ.

4.2　ESIGN 公開鍵

ESIGN 公開鍵は次の 3 つ組からなる.

- モジュラス n,　正の整数
- セキュリティパラメータ $pLen$,　正の整数
- 指数 e,　正の整数

正しい ESIGN 公開鍵　$2^{3pLen-1} < n < 2^{3pLen}$ が成り立つ.

5　プリミティブ

本章では, 3 種類のプリミティブを定義する.

5.1　KGP-ESIGN-TSH

KGP-ESIGN-TSH(k, e) を次のように定義する.

入力:　k　セキュリティパラメータ,　正の整数
　　e　公開される指数,　正の整数
出力:　PK　ESIGN 公開鍵 $(n, pLen, e)$
　　SK　ESIGN 秘密鍵 $(p, q, n, pLen, e)$
エラー:　`INVALID`
処理手順:

1. e が $e < 8$ なら, `INVALID` をエラー出力して処理を終了する.
2. $2^{k-1} < p < 2^k$, $2^{k-1} < q < 2^k$, $p \neq q$, $2^{3k-1} < p^2q < 2^{3k}$ を満たす奇素数 p, q を選び, $n := p^2q$ を計算する.
3. $pLen := k$ とする.
4. $PK = (n, pLen, e), SK = (p, q, n, pLen, e)$ を出力する.

5.2　SP-ESIGN-TSH

SP-ESIGN-TSH(SK, f) を, 次のように定義する.

入力: SK ESIGN 秘密鍵 $(p, q, n, pLen, e)$
f メッセージ, $0 \leq f < 2^{pLen-1}$ である整数
出力: s 署名, $0 \leq s < n$ である整数
エラー: `INVALID`
前提条件: 秘密鍵 SK は正しく生成されている
処理手順:

1. f が $0 \leq f < 2^{pLen-1}$ でなかったら, `INVALID` をエラー出力して処理を終了する.
2. $z := f \cdot 2^{2pLen}$ を計算する.
3. $\mathrm{GCD}(r, n) = 1$ を満たす $r \in \{1, 2, \ldots, pq-1\}$ をランダムに選ぶ.
4. $\alpha := (z - r^e) \bmod n$ を計算する.
5. (w_0, w_1) を以下のように計算する.

$$\begin{aligned} w_0 &:= \left\lceil \frac{\alpha}{pq} \right\rceil, \\ w_1 &:= w_0 pq - \alpha. \end{aligned}$$

6. w_1 が $w_1 \geq 2^{2pLen-1}$ なら, ステップ 3 へ戻る.
7. $t := w_0 (e r^{e-1})^{-1} \bmod p$ とし, $s := r + tpq$ を計算する.
8. s を出力する.

注意: 前提条件が成立していない場合, 本関数の動作は保証できない.

5.3 VP-ESIGN-TSH

VP-ESIGN-TSH(PK, s) を, 次のように定義する.

入力: PK ESIGN 公開鍵 $(n, pLen, e)$
s 署名, $0 \leq s < n$ である整数
出力: f 検証データ, $0 \leq f < 2^{pLen-1}$ である整数
エラー: `INVALID`
前提条件: 公開鍵 PK は正しく生成されている
処理手順:

1. s が $0 \leq s < n$ でなかったら, `INVALID` をエラー出力して処理を終了する.
2. $T := s^e \bmod n$ を計算する.
3. $f := \left\lfloor \frac{T}{2^{2pLen}} \right\rfloor$ を計算する.
4. f が $0 \leq f < 2^{pLen-1}$ でなかったら, `INVALID` をエラー出力して処理を終了する.
5. f を出力する.

注意: 前提条件が成立していない場合, 本関数の動作は保証できない.

6 署名スキーム

本章では, 1 種類の署名スキームを定義する.

6.1 SSA-ESIGN-TSH

SSA-ESIGN-TSH 署名スキームは, SP-ESIGN-TSH, VP-ESIGN-TSH プリミティブと EMSA-ESIGN-TSH エンコーディング処理を組み合わせたものである.

6.1.1 署名生成

SSA-ESIGN-TSH-Sign(SK, M) は, 次のように定義される.

入力: SK 署名者の ESIGN 秘密鍵
M 署名対象のメッセージ, オクテット列
出力: s 署名, $0 \leq s < n$ である整数
エラー: `INVALID`
処理手順:

1. EMSA-ESIGN-TSH-Encode 処理 (7.1.1 節) を用いて, メッセージ M を整数 f に変換する.

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

   もし, エンコード処理が `INVALID` というエラーを返した場合, `INVALID` をエラー出力して処理を終了する.

2. SP-ESIGN-TSH プリミティブ (5.2 節) を用いて, 秘密鍵 SK と整数 f から整数 s を得る.

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

3. 整数 s を署名として出力する.

6.1.2 署名検証

SSA-ESIGN-TSH-Verify(PK, M, s) は, 次のように定義される.

入力: PK 署名者の ESIGN 公開鍵
M メッセージ, オクテット列, これに対する署名が検証対象
s 検証対象となる署名, $0 \leq s < n$ である整数
出力: `VALID_SIGNATURE` または `INVALID_SIGNATURE`
処理手順:

1. VP-ESIGN-TSH プリミティブ (5.3 節) を用いて, 公開鍵 PK と整数 s から整数 f' を得る.

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

もし, プリミティブが INVALID というエラーを返した場合, INVALID_SIGNATURE を出力して処理を終了する.

2. EMSA-ESIGN-TSH-Verify 処理 (7.1.2 節) を用いて, f' が M の正しいエンコード結果であるかを調べる.

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

もし, *Result* が CONSISTENT ならば, VALID_SIGNATURE を出力する.
そうでなければ, INVALID_SIGNATURE を出力する.

7 エンコーディングメソッド

本章では, 1 種類のエンコーディングメソッドを定義する.

7.1 EMSA-ESIGN-TSH

以下のエンコーディングメソッドは, IEEE P1363a / D10 [1] の EMSA5 と互換性を持つ.

7.1.1 エンコーディング処理

EMSA-ESIGN-TSH-Encode(M, l) を, 次のように定義する.

システムパラメータ:	*Hash*	ハッシュ関数
	hLen	ハッシュ関数の出力ビット長, 正の整数
	MGF	マスク生成関数
入力:	M	エンコード対象のメッセージ, オクテット列
	l	出力ビット長, 正の整数
出力:	f	エンコード結果, 非負整数
エラー:	INVALID	

処理手順:

1. もし, M のビット長がハッシュ関数の入力ビット幅を越えていたならば, INVALID をエラー出力して処理を終了する.

2. M にハッシュ関数を作用させて, オクテット長 $\left\lceil \frac{hLen}{8} \right\rceil$ のハッシュ値 H を得る.

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

3. H にマスク生成関数を作用させて, オクテット長 $\left\lceil \frac{l}{8} \right\rceil$ のオクテット列 T を得る.

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

4. オクテット列 T を整数 f に変換する.

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

5. f を出力する.

7.1.2 検証処理

EMSA-ESIGN-TSH-VERIFY(M, f, l) を, 次のように定義する.

システムパラメータ:	$Hash$	ハッシュ関数
	$hLen$	ハッシュ関数の出力ビット長, 正の整数
	MGF	マスク生成関数
入力:	M	メッセージ, オクテット列
	f	検証データ, 非負整数
	l	f のビット長, 正の整数
出力:	CONSISTENT	
エラー:	INVALID	

処理手順:

1. 検証データ f を, ビット列 T へ変換する.

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

もし, I2BSP プリミティブが INVALID というエラー出力を返した場合, INVALID をエラー出力して処理を終了する.

2. もし, M のビット長がハッシュ関数の入力ビット幅を越えていたならば, INVALID をエラー出力して処理を終了する.

3. M にハッシュ関数を作用させて, オクテット長 $\left\lceil \frac{hLen}{8} \right\rceil$ のハッシュ値 H を得る.

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

4. H にマスク生成関数を作用させて, オクテット長 $\left\lceil \frac{l}{8} \right\rceil$ のオクテット列 T' を得る.

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

5. オクテット列 T' をビット列 T'' に変換する.

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

6. もし, T と T'' が同じならば, CONSISTENT を出力する. そうでなければ, INVALID をエラー出力する.

8 補助関数

本章では, 本ドキュメントのさまざまな定義式に用いられる関数を説明する.

8.1 ハッシュ関数

本ドキュメントでは, SHA-1 ハッシュ関数を推奨する.

8.1.1 SHA-1

SHA-1 は FIPS PUB 180-1 [2] で定義されている. SHA-1 の出力長は 160 ビットであり, 処理ブロックサイズは 512 ビットである.

8.2 マスク生成関数

本ドキュメントのエンコーディングメソッドでは, MGF1 マスク生成関数 [3] を推奨する.

8.2.1 MGF1

MGF1 はハッシュ関数を利用したマスク生成関数である.
MGF1(M, l) は次のように定義される.

システムパラメータ:	$Hash$	ハッシュ関数
	$hashLen$	ハッシュ関数の出力ビット長, 正の整数
入力:	M	メッセージ, オクテット列
	l	出力ビット長, 正の整数
出力:	$mask$	マスク, 長さ $\left\lceil \frac{l}{8} \right\rceil$ のオクテット列
エラー:	`INVALID`	

処理手順:

1. l_0 を M のビット長とする. もし, $l_0 + 32$ がハッシュ関数の入力ビット幅を越えていたならば, `INVALID` をエラー出力して処理を終了する.
2. $cThreshold := \left\lceil \frac{l}{hashLen} \right\rceil$ を計算する.
3. M' を空オクテット列とする.
4. $counter := 0$ とする.
   (a) $counter$ を 32 ビット長のオクテット列 C に変換する.
   $$C := \mathrm{I2OSP}(counter, 32).$$
   (b) M と C を連結したものにハッシュ関数を作用させて, オクテット長 $\left\lceil \frac{hashLen}{8} \right\rceil$ のハッシュ値 H を得る.
   $$H := Hash(M \,\|\, C).$$
   (c) M' と H を連結し, その結果を再び M' とする.
   $$M' := M' \,\|\, H.$$

(d) *counter* を 1 増やす. もし $counter < cThreshold$ ならば, ステップ 4a に戻る.

5. M' の先頭 $\left\lceil \frac{l}{8} \right\rceil$ オクテットを *mask* とする.

$$mask := M'_0 M'_1 \cdots M'_{\lceil l/8 \rceil - 1}.$$

6. *mask* を出力する.

参考文献

[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.

付録

A　パラメータ設定基準

ESIGN-TSH パラメータの設定基準は以下の通りである.

$$\begin{aligned} k &\geq 342 \qquad (n \text{ のビット長は } 1024 \text{ 以上}) \\ e &\geq 8 \end{aligned}$$

B　パラメータ推奨値

ESIGN-TSH パラメータの推奨値は以下の通りである.

$$\begin{aligned} k &= 384 \qquad (n \text{ のビット長は } 1152) \\ e &= 1024 \\ Hash &= \text{SHA-1} \\ hLen &= 160 \\ MGF &= \text{MGF1(SHA-1, } hashLen = 160) \end{aligned}$$

C　推奨利用法

ESIGN-TSH については, メッセージ M に対して 1 回のみ署名されるような利用のされ方を推奨する。これは、ESIGN-TSH が CMA ではなく OM-CMA に対して安全であることを保証しているためである (??? における ESIGN 自己評価書 Theorem 2.3 を参照).

よって, 署名対象となるメッセージ M_0 には, 署名者が独自に定める連番, 日付, 乱数 など, 1 回しか用いられないようなデータ OT が付加されることが望ましい. すなわち,

$$M := M_0 \,\|\, OT \qquad (OT \text{ は 1 回しか用いられないオクテットストリング}).$$

OT の例:

- 以下のような 20 オクテットのオクテット列

$$\begin{aligned} OT &= \text{I2OSP}(c, 160). \\ c &: \text{署名者が付与する連番, } 0 \leq c < 2^{160}. \end{aligned}$$

- 以下のような 20 オクテットのオクテット列

$$\begin{aligned} OT &= \text{I2OSP}(ps, 160). \\ ps &: \text{1970 年 1 月 1 日 00:00:00.000000000000 UTC からの時刻 (ピコ秒),} \\ &\quad 0 \leq ps < 2^{160}. \end{aligned}$$

- 以下のような 20 オクテットのオクテット列

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