2008 年度版リストガイド
（電子署名）

平成 21 年 3 月

独立行政法人情報通信研究機構
独立行政法人情報処理推進機構

目 次

1　電子署名

1.1　本文書の位置付け

1.1.1　文書の目的

通信データや蓄積データの偽造、改ざん、破損等を検知し、データの完全性を保証することは電子政府システムにおいて非常に重要である。公開鍵暗号系の技術を応用し、秘密鍵を用いて認証子を計算し、公開鍵を用いて認証子を検査することで完全性のチェックを行うメカニズムは、電子署名 (Digital Signature) とよばれる。

本リストガイドでは、FIPS 186-3 Draft [FIPS186-3] 記載の (楕円) 離散対数問題ベースの電子署名技術である DSA, ECDSA、および RSA PKCS#1 v.2.1 [PKCS1] 記載の素因数分解問題 (RSA 問題) ベースの RSASSA-PKCS1-v1_5, RSASSA-PSS の技術概要、および実装仕様を記述する。

1.1.2　対象とする利用目的

電子署名は、相手認証、データの完全性保証に利用可能である。公開鍵暗号系の技術を応用し、認証局により正当性が保証された公開鍵を被保護の通信路で共有して検証処理を行うため、MAC と異なり事前に鍵 (秘密鍵) を秘密裏に共有する必要がない。

1.1.3　本文書の構成

本文書は、以下からなる。1.2 節で、各電子署名技術に共通して用いる用語および略称を記述する。記号および記法は、参照元の標準文書により異なる意味で用いられることがあるため、1.4 節で電子署名技術ごとに記述する。1.3 節で、電子署名の技術概要、および本文書に記述する DSA，ECDSA，RSASSA-PKCS1-v1_5，RSASSA-PSS の概要を示す。1.4 節では、個々の電子署名技術の実装仕様を記述する。

1.2　用語および略称

本文書で用いる用語および略称を記述する。記号および記法は、参照元の標準文書により、同一の記号が異なる意味で用いられることがある。そのため、1.4 節で個々の実装仕様を記述する際にまとめる。

メッセージ　署名者が署名生成を行う対象の文書。

署名者　電子署名を生成するエンティティ。

検証者　電子署名を検証するエンティティ。

公開鍵認証基盤 (PKI)　公開鍵証明書を発行、保守、失効する機能を実現するための枠組み。

認証局 (CA)　公開鍵認証基盤 (PKI) において、公開鍵証明書の発行に責任を持ち、PKI ポリシーを厳格に運営するエンティティ。

公開鍵証明書　（データには含まれていないが秘密鍵と）公開鍵の鍵ペアと、その鍵ペアを用いる権限が与えられたユーザを一意に特定するデータの集合。証明書は、所有者の公開鍵と、他の関連するデータを含み、それらへの CA（信頼できる第三者機関）によって署名されたデータを含んでいる。これにより、公開鍵とその所有者が関連付けられる。

ビット　Bit。2 進の数。0 または 1。

オクテット　Octet。8 ビットの数。16 進表記の 0x00 から 0xff。

ビット列　0 と 1 からなる順序付けられた列。最も左のビットは最高位ビットであり、最も右のビットは最下位ビットを表す。

電子署名　データの作成者の認証、データの完全性、署名者の否認不可性を検証する機能を実現する、データの暗号学的な変換の結果。署名対象の文書 (データ) に、以下に記述する署名生成鍵を作用させて得られる。

秘密鍵　署名者が署名生成に用いるために秘密裏に保持すべきビット列。

署名生成鍵　署名者が電子署名を作成するときに用いるビット列。秘密鍵のほかに、公開パラメータを含むこともある。

公開鍵　秘密鍵に対応するビット列。

署名検証鍵　検証者が電子署名の正当性を検証するときに、署名対象文書、電子署名と共に用いるビット列。公開鍵と同じ意味で用いる。

ANS　American National Standard。

CA　Certification Authority。

CRYPTREC　Cryptography Research and Evaluation Committees。

DSA　Digital Signature Algorithm (specified in this Standard)。

ECDSA　Elliptic Curve Digital Signature Algorithm specified in ANS X9.62。

FIPS　Federal Information Processing Standard。

NIST　National Institute of Standards and Technology。

PKCS　Public Key Cryptography Standard。

PKI　Public Key Infrastructure。

PSS　Probabilistic Signature Scheme。

RBG Random Bit Generator; specified in SP 800-90。

RSA Algorithm developed by Rivest, Shamir and Adelman (specified in ANS X9.31 and PKCS #1)。

SHA Secure Hash Algorithm; specified in FIPS 180-3。

SP NIST Special Publication。

TTP Trusted Third Party。

1.3 技術概要

1.3.1 暗号技術の利用モデル

電子署名技術では、電子署名を生成する署名者、署名の正当性を検証する検証者の 2 つのエンティティが存在する。必要に応じて、署名者の公開鍵の正当性を保証する第三者 (認証局) が用いられる。署名者と検証者はそれぞれ署名生成装置と署名検証装置を保持する。

署名生成装置は、秘密鍵を含む署名生成鍵 sk を秘密裏に保持し、入力された署名対象文書 M に対して署名 S を生成し、M と S を出力する。このとき、sk および対応する署名検証鍵 (公開鍵) pk は、署名生成装置が予め生成してもよいし、署名者が別に保持する署名生成・検証鍵生成装置で生成してもよい。

検証者は、署名者から M、S、署名者の署名検証鍵 (公開鍵) pk を入手する。認証局を用いる場合には、認証局が発行する公開鍵証明書とともに pk を入手する。

署名検証装置は、M、S、pk を入力として、S が M、pk に対して正当であると判定するときには VALID を出力し、正当でないと判定するときには INVALID を出力する。

1.3.2 技術の基本的構成

電子署名の技術の基本的構成として、鍵生成、署名生成、署名検証がある。これらを実行するアルゴリズムについて説明する。

鍵生成アルゴリズム 鍵生成アルゴリズムは、署名者の署名生成鍵と署名検証鍵を生成するアルゴリズムである。鍵生成アルゴリズムは、署名者により実行される確率的アルゴリズム[1]であり、安全性のパラメータ k を入力として、署名者の署名生成鍵 sk と署名検証鍵 pk を出力する。署名者は sk を安全に秘密裏に管理し[2]、pk を (必要に応じて認証局が発行する証明書と共に) 公開する。

[1]確率的アルゴリズムとは、内部で選択される乱数により、同一の入力に対しても出力が一意に定まらないアルゴリズムをいう。

[2]厳密には、署名生成鍵 sk のうち、公開パラメータを除く秘密鍵を安全に秘密裏に管理する。

署名生成アルゴリズム　署名生成アルゴリズムは、署名者が署名を生成するために用いるアルゴリズムであり、署名対象文書 M と sk を入力として、署名 S を出力する。署名生成アルゴリズムは、確定的アルゴリズム[3]であっても確率的アルゴリズムであってもよい。

署名検証アルゴリズム　署名検証アルゴリズムは、検証者が署名を検証するために用いるアルゴリズムであり、M、S、pk を入力として、S が M と pk に対して正当な署名であると判定する場合には VALID を出力し、正当な署名ではないと判定する場合には INVALID を出力するアルゴリズムである。

電子署名技術の基本的な機能は、署名対象文書に対して署名者が生成した署名を検証することにより、“署名者” が “署名対象文書の内容を承認した” ことを、検証者が確認することである。そのため、セキュリティ要件として、適応的文書攻撃に対して存在的偽造困難 [GMR84] であることが求められる。

適応的文書攻撃に対して存在的偽造困難であるとは、直観的には、電子署名技術を破ろうとする攻撃者が、署名生成装置を自由に利用することができたとしても、署名生成装置が署名を発行していない文書に関する正当な署名を偽造することができないことをいう。このとき、攻撃者は公開されている pk も利用できる。例えば、署名生成サーバの実装に不備があり、不備をついていくつかの文書に対する署名をサーバから入手したとしても、それらをヒントに新たな文書に対する署名を偽造できない。

技術的には、以下の手続きで動作する多項式時間攻撃者[4]の成功確率が無視できるほど小さいことをいう。

1. 攻撃者は、安全性のパラメータ k と公開鍵 pk を入手する。
2. 攻撃者は、文書 M_i を署名生成装置[5] に問い合わせ、署名 S_i を入手する。ここで i は、回数をあらわすインデックスであり、$1 \leq i \leq q_S$ をみたす。ただし、q_S は署名生成装置を利用することができる回数の上限である。
3. 攻撃者は、M, S を出力する。

ステップ 3 で出力された M, S について、$M \notin \{M_i\}_i$ かつ、署名検証装置が M, S, pk を入力として 1 を出力するとき、攻撃者は適応的文書攻撃に対して存在的な偽造に成功するとよぶ。

1.3.3　評価観点と比較

RSASSA-PKCS1-v1_5　RSASSA-PKCS1-v1_5 は、RSA 問題の計算困難性に安全性の根拠をおく確定的な署名方式であるが、安全性証明は与えられていない。パラメータの選択や、実装の不備を利用する解読や攻撃が提案されており、RSASSA-PSS への移行が進められている。既存システムとの整合性をとる目的以外では使用されるべきではない。

---

[3]確定的アルゴリズムとは、入力に対して出力が一意に定まるアルゴリズムをいう。

[4]多項式時間攻撃者とは、安全性のパラメータ k に対して動作時間が多項式時間に制約される攻撃者をいう。この制約を外すと、計算量的仮定に基づく方式の安全性は保証できない。例えば、安全性のパラメータに対して指数時間の動作が許されている攻撃者は、公開鍵から対応する秘密鍵を計算することができるかもしれない。

[5]安全性の概念の定式化においては、署名生成オラクル (*signing oracle*) とよばれる。

RSASSA-PSS　RSASSA-PSS は、RSA 問題の計算困難性に安全性の根拠をおく確率的な署名方式であり、ランダムオラクルモデルの下で安全性証明が与えられている。

ランダムオラクル [BR93] とは、初出の入力に対しては出力がランダムに決定され、既知の入力に対しては過去に決定された値を出力する理想的な関数のことをいう。

RSA 問題とは、RSA の署名検証鍵 (n, e) と、ある $y \in \mathbb{Z}_n$ が与えられたときに、$y^(1/e) \bmod n$ を求める問題をいう。

表 1 に、それぞれの方式の安全性 (安全性証明の有無、有の場合に仮定する問題および安全性の根拠)、推奨パラメータをまとめる。

表 1: 署名方式の比較

署名アルゴリズム名			DSA	ECDSA	RSASSA-PKCS1-v1_5	RSASSA-PSS
参照すべき仕様					RSA-PKCS#1 v2.1	RSA-PKCS#1 v2.1
安全性	経験的安全性 (注 1)		○	○	○	○
	証明可能安全性		—	—	—	○
		仮定するモデル	—	—	—	ランダムオラクルモデル
		帰着される問題	—	—	—	RSA 問題の計算困難性
		安全性の到達度 (注 2)	—	—	—	適応的選択文書攻撃に対して存在的偽造困難
プリミティブの安全性の根拠			有限体上の離散対数問題	楕円曲線上の離散対数問題	素因数分解問題	素因数分解問題
パラメータ	推奨鍵長 (注 3)		2048, 3072	224, 256, 384, 512	2048, 3072	2048, 3072
	ハッシュ関数		電子政府推奨暗号リストに記載されたものを使用すること (注 4)			
	擬似乱数生成器		電子政府推奨暗号リストの注釈 7 を参照のこと			

(注 1) アルゴリズムが経験的に安全であるとは、アルゴリズムに対して具体的な攻撃手順が知られていないことをあらわす。
(注 2) 安全性の到達度とは、1.3.2 節で述べたように、攻撃者の能力と攻撃目的を定式化し、安全性証明を記述しうる最も高い安全性のモデルをあらわす。
(注 3) 推奨鍵長は CRYPTREC の推奨値を記述する。
(注 4)DSA, ECDSA はハッシュ関数として SHA-1 を利用する。

1.3.4 参考情報

RSA-PKCS#1 v2.1 では、RSASSA-PKCS1-v1_5 と RSASSA-PSS の署名検証鍵で用いる法 n として 3 つ以上の奇素数の合成数を許すとしている。本文書では、一般的に用いられる、n として 2 つの奇素数の合成数を用いる場合を説明する。n として 3 つ以上の奇素数の合成数を用いる場合には、安全性を保証するために n を大きくとらなければならないため、計算量が増大する。

NIST FIPS 186-3 Draft (2008 年 12 月発行) では、以下の 6 点が制約されている。

- 鍵組の生成には同ドラフトの付録 B.3.1 の指針と付録 B.3.2 の手法を用いることとする。
- 認証されたハッシュ関数のみを利用することとする。
- n の生成には p, q の二つの素数のみを用いることとする。
- 乱数の生成は SP800-90 に従うことする。
- RSASSA-PSS について、salt の長さは $0 \leq sLen \leq hLen$ (ハッシュの出力長) とする。
- 署名検証時において、エンコード EM (Encoded Message) がフォーマットどおりに作成されていることを検査する。

上の 6 点目について，NIST FIPS 186-3 Draft(2008 年 12 月発行) では，Bleichenbacher の攻撃 [B06] を防ぐことを目的として,

- RSASSA-PKCS1-v1_5 について、ハッシュ値を EM (Encoded Message) から復元するときには、以下のどちらかを満たすこととする：
  - エンコードの長さに関わらず、EM の最下位ビットから選択する
  - エンコードの長さに依存してハッシュ値の位置を選択する場合には、EM の最下位バイトを含むことを確認する

と述べている。しかし、大岩ら [OKW07] が示すように、ハッシュ値が最下位バイトを含む場合でも、EM の中間部分がフォーマットと異なることを許す場合には署名の偽造が可能となる。そのため、EM の全フォーマットを適切に検査する必要がある。

DSA は米国 NIST によって提案、標準化された電子署名方式であり、電子署名法に係る指針に記載されている。NIST は現在、FIPS 186-2 を改訂作業中で、FIPS 186-3 Draft では、パラメータ p の大きさを、1024, 2048, 3072 ビットから選択可能とし、ハッシュ関数についても、SHA-1 だけでなく、SHA-224 および SHA-256 を利用可能としている。安全性は有限体上の離散対数問題の困難性に依存している。証明可能安全性は示されていないが経験的に安全である (表１の (注釈１) 参照)。CRYPTREC では、その参照仕様として、FIPS 186-2 (+Change Notice 1) を使用している。その一方で、CRYPTREC では安全性の観点から、パラメータ p のサイズは 2048 ビット以上を選択すること、ハッシュ関数として SHA-1 を使用しないことが推奨されている。本稿では、これから実装する際に一番参考となると考えら

れる、FIPS 186-3 Draft をベースに説明を行う。なお、この文書には FIPS 186-2 (+Change Notice 1) で定義されていた DSA が含まれている。

ECDSA は、ANS X9.62, FIPS 186-3 Draft, SEC 1 などで標準として規定されている。ECDSA の安全性は楕円曲線上の離散対数問題の困難性に依存している。証明可能安全性は示されていないが経験的に安全であり、安全性について重大な問題点は指摘されていない。鍵生成プロセスに十分に留意すれば安全性に問題はないと考えられる。ECDSA の ANS X9.62 版と ECDSA の SEC 1 版はスキームとして同一であり、規定される楕円曲線パラメータについてもほぼ同じである。FIPS 186-3 Draft 版も ANS X9.62 版をもとに規定されている。本稿では以降 DSA との記述の統一も考え、FIPS 186-3 Draft をベースに説明を行い、必要に応じて ANS X9.62 版との違いを述べる。

1.4 実装仕様

1.4.1 DSA と ECDSA で用いる記号の定義

$a \mod n = r$	:	整数 a が n で除算された際の $0 \leq r \leq (n-1)$ なる剰余 r
$counter$	:	DSA で、ドメインパラメータの種を用いてドメインパラメータを生成した結果で定まるカウンタ値。
d	:	ECDSA での署名生成用の秘密鍵。
$domain\_parameter\_seed$	:	ドメインパラメータを生成するための種。
g	:	DSA のドメインパラメータ。g はオーダ q の巡回群 $\mathrm{GF}(p)^*$ の生成元。
$\mathrm{Hash}(M)$	:	FIPS 186-3 Draft においては、FIPS 180-3 で規定されたハッシュ関数を用いて計算した結果。
$index$	:	g の生成時に、それがどのような用途で用いられるか（この場合は電子署名）を示す値。
k	:	DSA と ECDSA においては、署名生成ごとに生成される秘密情報。
L	:	DSA においては、パラメータ p のビット長。
(L, N)	:	DSA の鍵ペアに対する長さのパラメータ対。L は p の長さを、N は q の長さを表す。
M	:	電子署名を施す対象メッセージ。
m	:	ECDSA において、有限体 GF_{2^m} の次数。
N	:	DSA において、q のビット長。
n	:	ECDSA において、使用する楕円曲線のベースポイントの次数。n のビット長が鍵のサイズとみなされる。
p	:	DSA において、ドメインパラメータの 1 つ。$GF(p)$ を定義する素数であり、この体での剰余演算を行う際に用いられる。
q	:	DSA において、ドメインパラメータの 1 つ。$p-1$ の素約数。
Q	:	ECDSA において、署名検証用の公開鍵。
(r, s)	:	DSA あるいは ECDSA の署名データ。r, s はそれぞれ署名データの要素。
x	:	DSA において、署名生成用の秘密鍵。
y	:	DSA において、署名検証用の公開鍵。

1.4.2 DSA

パラメータ

DSA では、ドメインパラメータ、秘密鍵 x、署名ごとに使用する秘密の情報 k、署名対象となるメッセージ、ハッシュ関数を用いて署名データを計算する。一方、署名データを検証する際には、同一のドメインパラメータ、署名データ生成時に使用された秘密鍵と（検証処理が正当に働くよう）、数学的に関連付けられている公開鍵 y、検証対象となる署名データ、署名データ生成時に使用したハッシュ関数を用いる。これらのパラメータは以下のように定義される。

p : $2^{L-1} < p < 2^L$ なる素数である法。L は p のビット長。
L の値は次の「パラメータサイズとハッシュ関数の選択」で与えられる。

q : $2^{N-1} < q < 2^N$ なる $(p-1)$ の素約数。N は q のビット長。
N の値は次の「パラメータサイズとハッシュ関数の選択」で与えられる。

g : $1 < g < p$ なる位数 $q \mod p$ の部分群の生成元。

x : $0 < x < q$ すなわち区間 $[1, q-1]$ から（擬似）ランダムに選ばれ、
秘密に保持されなければならない秘密鍵。

y : $y = g^x \mod p$ なる公開鍵。

k : $0 < k < q$ すなわち区間 $[1, q-1]$ から（擬似）ランダムに選ばれ、
各署名生成時に生成される秘密の情報。

パラメータサイズとハッシュ関数の選択

FIPS 186-3 Draft では、L, N の組（p, q のビット長）について、以下のような候補を定義しており、これらの中から選ぶことを必須としている。

$L = 1024, \quad N = 160$
$L = 2048, \quad N = 224$
$L = 2048, \quad N = 256$
$L = 3072, \quad N = 256$

DSA において、署名生成時にはハッシュ関数が必要となる。用いることができるハッシュ関数は FIPS 180-3 規定のものから選ぶことを必須としている。ちなみに CRYPTREC では、新たなシステムを構築する際には 256 ビット以上のハッシュ関数を用いることが望ましいとされている。ハッシュ関数を選択する際、(L, N) の組で決まる安全性よりも高い安全性を持つものを選択することが必須とされている。これら安全性については、SP 800-57 で述べられている[6]。FIPS 186-3 Draft では、(L, N) の組の安全性とハッシュ関数の安全性を同じとすることが推奨されている。安全性の低いハッシュ関数を使用することは禁止されている。N ビットよりも長いハッシュ関数を用いる場合、その出力の左 N ビットを使用して演算することが必須とされている。

SP 800-57 には、2008 年、2010 年といった各時点における、これらビット長と安全性の関係が述べられている。CA が使用すべき (L, N) の組の指針についても同ドキュメントに書かれているのでそちらを参照のこと。

ドメインパラメータ

DSA では、ドメインパラメータと呼ばれる複数のデータに基づいて、秘密鍵と公開鍵が生成され、それを用いて署名生成や署名検証が行われる。ドメインパラメータは、あるユーザグループに共通で使用してもよく、公開としてもよいものである（当然、複数のユーザグループがある場合、それぞれのグループに対応した複数のドメインパラメータが規定されている場合も存在する）。署名者と検証者は、ドメインパラメータについて、それらを用いる前に、それらの値の正当性に関して何らかの

[6] $L = 2048, N = 256$ の組は、SP 800-57 に掲載されていないが、L, N それぞれの値を用いた組の安全性のうち、低いほうの安全性で見積もれば、安全上の問題はない。この場合、$L = 2048, N = 224$ が 112 ビットの安全性、$L = 2048, N = 224$ が 128 ビットの安全性であるので、$L = 2048, N = 256$ は 112 ビットの安全性を持つと考えればよい。

保証を得ていることが必須となっている。したがって、ドメインパラメータは公開してもよいことになっているが、秘密鍵／公開鍵の組とドメインパラメータとの関連付けは適切に管理されていることが必須となる。

DSA におけるドメインパラメータは、整数 p, q, g と、オプションとして p, q を生成する際に使用される $domain\_parameter\_seed, counter$ がある（すなわち $(p, q, g, \{domain\_parameter\_seed, counter\})$ のように表される）。

ドメインパラメータの生成　不特定多数の間で通信が行われる環境においては、一般にドメインパラメータは、CA のような信頼できる第三者機関 (TTP: Trusted Third Party) によって生成される。ドメインパラメータの正当性を確認することを、秘密鍵／公開鍵の組を生成し、それらを用いて署名生成、署名検証を行う前に行うことは必須とされている。

FIPS 186-3 Draft では、素数 p, q は、同 Appendix A.1 で規定された素数生成法（素数候補生成法とその素数検証法）を用いて生成することが必須とされている。詳細はそちらを参照のこと。入力値として L, N を与え、出力値として $domain\_parameter\_seed$, $conter$ を得る。生成元 g の計算も、同 Appendix A.2 で規定された方法で計算することが必須とされている。

ドメインパラメータの管理　署名で用いられる秘密鍵と公開鍵の組は、対応するドメインパラメータと適切に関連付けられて管理されることが必須とされている。ドメインパラメータは、それが有効な期間、不適切な変更から保護されることが必須とされている。

秘密鍵と公開鍵

各署名者は、秘密鍵 x と、それと（署名生成と署名検証が適切に行えるよう）数学的に関連付けられた公開鍵 y の組を持っている。秘密鍵は定められたある期間、すなわち署名生成を行ってよいとされている期間（安全性が保証されている期間）でのみ、使用を認めることを必須としている。一方で公開鍵は、対応する秘密鍵で署名されたデータを検証することが必要とされる間は、署名検証に使用してよい。これら鍵の利用方法の指針は SP 800-57 に述べられているのでそちらを参照のこと。

鍵生成アルゴリズム　秘密鍵と公開鍵の組 x, y は、ドメインパラメータ $(p, q, g$ $\{, domain\_parameter\_seed, counter\})$ から生成される。生成法は FIPS 186-3 Draft Appendix B.1 に規定されており、それを用いることになっている。

鍵管理　鍵管理の指針は SP 800-57 に書かれている。電子署名の安全な利用は、エンティティでの適切な鍵管理による。FIPS 186-3 Draft では、以下は必須とされている。

1. ドメインパラメータの正当性をあらかじめ確認した上で、秘密鍵／公開鍵の組の生成、署名生成、検証処理を行う。
2. 鍵の組とそれを生成する際に使用したドメインパラメータは、適切に関連付けられるようになっている。

3. 鍵の組は、その組と適切に関連付けられたドメインパラメータとを用いてのみ使用する。

4. 秘密鍵は、FIPS 186-3 Draft で規定された署名生成においてのみ使用し秘密に保持する。公開鍵は、同様に署名検証においてのみ使用する。公開鍵は公開情報としてよい。

5. 署名者は、署名生成を行う以前、あるいは行う時点で、署名生成に用いる正しい自分の秘密鍵を保持していることを確認済みである。その確認方法については SP 800-89 に述べられている。

6. 秘密鍵は、不正なアクセス、暴露、変更から保護されている。

7. 公開鍵は不正な（置き換えを含む）変更から保護されている。例えば、ある CA によって署名された公開鍵証明書によって、これは実現できる。

8. 検証者は、公開鍵、対応づけられたドメインパラメータ、鍵の所有者の 3 つの間の関連の正当性が確認できる。

9. 検証者は、公開鍵を信頼のおける方法で得ることができる。例えば、検証者が信頼している CA によって署名された公開鍵証明書を利用して、あるいは、当該署名者であると主張している当人から公開鍵が送付され、自身が検証者に信頼されていることと、検証対象である署名データの送信者であることが認証されていることを示すこと、などでこれは実現できる。

10. 検証者は、署名者であると主張しているエンティティが検証で使用する鍵の組の所有者であり、検証しようとしている署名データを生成した時点で、それに用いた秘密鍵を所有していたかどうかを確認できる。

11. 署名者と検証者はそれぞれ、処理で用いられる公開鍵の正当性を確認済みである。

署名生成時に使用する秘密情報

DSA では署名生成処理ごとに、署名データ生成前の時点で、新たに秘密の情報 k を（擬似）ランダムに生成する。FIPS 186-3 Draft では必須とされている。k は不正な暴露や変更から保護されることも必須とされている。

k^{-1} は、q を法とする乗算に関する $0 < k^{-1} < q$ かつ $1 = (k^{-1}k) \mod q$ なる k の乗法的逆元である。この逆元は、署名生成処理において必要となる情報である。k から k^{-1} を計算する方法については、FIPS 186-3 Draft Appendix B.2 に規定されている。

k と k^{-1} は、署名されるメッセージの乗法を必要としないため、事前に計算しておいてもよい。これらが事前計算されている場合、これらの秘匿性と完全性を保証することが必須とされている。k の計算方法については、FIPS 186-3 Draft Appendix B.2 に規定されている。

署名生成アルゴリズム

N を q のビット長であるとする。メッセージ M の署名データは、以下のように計算された r, s の組から構成される：

$$\begin{aligned} r &= (g^k \mod p) \mod q \\ z &= \text{Hash}(M) \text{の左から} N \text{ビットのビット列} \\ s &= (k^{-1}(z + xr)) \mod q \end{aligned}$$

s を計算する際、Hash(M) の結果のビット列から整数値へと変換する処理が必要となる。その方法については Appendix C.2 に規定されている。

ここで留意すべきは、r は k, p, q があればいつでも計算できるということである。例えば、ドメインパラメータである p, q が既知であり、k が事前計算されていた場合、r も事前計算可能となる。このとき事前計算されている k, k^{-1}, r は、秘密鍵 x と同様の方法で、s が計算されるまで保護されることが必須となっている（詳しくは SP 800-57 を参照)。

r と s に関して、$r = 0$ あるいは $s = 0$ となるかどうかを検査することが必須とされている。もしどちらかでも 0 であった場合は、新たに k を生成して処理をやり直すことが必須とされている。実際には、そのようなことはほぼ起こらないと言ってよい。

署名データ (r, s) は、メッセージと一緒に検証者へ送付される。

署名検証アルゴリズム

署名検証は、署名者の公開鍵を用いて誰が行ってもよい。署名者は、生成した署名データが正しいことを、意図する検証者に送る前に自身で確認してもよい。署名検証者（あるいは他のエンティティ）は、署名データの真正性を検証する。

あるメッセージの署名データを検証する前に、ドメインパラメータと署名者の公開鍵、署名者の ID の 3 つは、検証者が正当性を確認できる形で、利用可能となっていることが必須とされている。公開鍵は例えば、CA のような信頼できる第三者によって署名された公開鍵証明書を用いて、あるいは直接公開鍵の所有者から手渡しされる等の方法で得ることができる。

M', r', s' をそれぞれ M, r, s に対応する受信データであったとし、y を署名者であると主張しているエンティティの公開鍵であるとする。また、N を q のビット長であるとする。このとき、署名検証処理は以下のようになる：

1. 検証者は、$0 < r' < q, 0 < s' < q$ かどうかを検証する。満たさない場合は無効な署名であるとして、「INVALID」を出力して処理を終了する。この処理は必須とされている。

2. 検証者は、次のような計算を行う。

$$\begin{aligned} w &= (s')^{-1} \mod q. \\ z &= \text{Hash}(M') \text{の左から} N \text{ビット}. \\ u_1 &= (zw) \mod q. \\ u_2 &= ((r')w) \mod q. \\ v &= (((g)^{u_1}(y)^{u_2}) \mod p) \mod q. \end{aligned}$$

$(s')^{-1}$（すなわち s' mod q の乗法的逆元）の計算方法については、Appendix C.1 に規定されている。

3. もし、$v = r'$ であれば、署名検証が成功したことになる。このとき、「VALID」を出力して終了する。$M' = M, r' = r, s' = s$ であるとき、$v = r'$ が成立することの証明は、Appendix E に書かれている。

4. もし、$v = r'$ でない場合、メッセージあるいは署名データが変更されている可能性、署名者が署名生成を失敗した可能性、秘密鍵を知らない不正な署名者が署名偽造を企てて送りつけた可能性、が考えられる。このとき、署名を無効とすることが必須とされている。署名データが正当なものかどうかについては、署名を検証する公開鍵を使用する際、署名データがそのメッセージデータに対して適切なデータでないこと以外、全く推測されない。

5. 署名データを正当なものとして受理する前に、検証者は (1) 署名者が主張する ID の正当性、(2) ドメインパラメータの正当性、(3) 公開鍵の正当性、(4) 署名者が、署名データが生成された際に、署名を作るために使用された秘密鍵を保持していたことを確認できることが必須とされている。

1.4.3 ECDSA

ドメインパラメータ

ECDSA では、署名生成と署名検証でそれぞれ用いられる、秘密鍵と公開鍵がある特定のドメインパラメータのもとで生成される。ドメインパラメータは、ユーザグループで共通に用いられ、公開される。ドメインパラメータは署名生成されなくなった後でも有効である場合も存在する。

ECDSA のドメインパラメータは

$$(q, FR, a, b, domain\_parameter\_seed, G, n, h)$$

ここで

q	有限体のサイズ。
FR	使用している基底の識別子。
a と b	2 つの体の元であり曲線を定義する要素。
$domain\_parameter\_seed$	ドメインパラメータを生成する種と、楕円曲線がランダムであることが検証できる形で生成された場合に与えられるビット列。
G	曲線上の素位数のベースポイント（$G = (x_G, y_G)$）。
n	点 G の位数。
h	余因子 (cofactor)、すなわち曲線の位数を n で割ったもの。

の形で与えられる。

表 2: ECDSA のセキュリティパラメータ

n のビット長 $\lceil \log_2 n \rceil$	最大の余因子 (h)
$160 - 223$	2^{10}
$224 - 255$	2^{14}
$256 - 383$	2^{16}
$384 - 511$	2^{24}
≥ 512	2^{32}

ドメインパラメータの生成　FIPS 186-3 Draft では以下表のように、n の範囲を 5 つ定めている。それぞれの範囲において、最大の余因子サイズが定められている。なお ANS X9.62 では、ドメインパラメータのうち、余因子 h の仕様は オプションとなっている。

FIPS 186-3 Draft では、素体 GF_p（p は奇素数）と 2 の拡大体 GF_{2^m} 上で ECDSA を定義している。同 Appendix D で以下の 3 種類の曲線が与えられている。

1. 素体上の曲線（P_{-XXX} で表記される）
2. 2 の拡大体上の曲線（B_{-XXX} で表記される）
3. Koblitz 曲線（K_{-XXX} で表記される）

ここで、XXX は体のサイズのビット長を表している。

FIPS 186-3 Draft では、ドメインパラメータの生成に ANS X9.62 の方法を用いても良いことになっている。ANS X9.62 では、G は canonical に生成することが推奨されている。ANS X9.62 G の canonical な生成では、*element* と呼ばれる値を 1 に設定することが求められている。FIPS 186-3 Draft では、楕円曲線のドメインパラメータが、鍵共有や乱数生成等の他の目的で生成される場合、*element* は異なる値を用いることが推奨されている。結果として、それぞれの機能で異なる G が用いられることになる。

パラメータサイズとハッシュ関数の選択　ECDSA（FIPS 186-3 Draft 版）において、署名生成時にはハッシュ関数が必要となる。用いることができるハッシュ関数 Hash は FIPS 180-3 規定のものから選ぶことを必須としている（一方、現在の CRYPTREC 参照先となっている FIPS 186-2 (Change Notice 1) 版では SHA-1 に限定）。ちなみに CRYPTREC では、新たなシステム構築する際には 256 ビット以上のハッシュ関数を用いることが望ましいとされている。使用するハッシュ関数の安全性の強度は、署名利用で求められている安全性の強度を表すビット長と同等以上を持つものであることが推奨されている。n が表す範囲の安全性の強度とハッシュ関数の安全性の強度の関係については、SP 800-57 に述べられている。n のビット長と関連付けられた安全性の強度と、ハッシュ関数の安全性の強度を同じにすることが推奨されている。一方で、ハッシュ関数の安全性強度の方が低い場合、そのハッシュ関数を使わないことが推奨されている。ハッシュ関数の出力が n のビット長よりも長い場合、出力の左から n のビット長分を取り出して、署名生成や署名検証に必要な処理に用いることが推奨されている。

一般に CA は、署名者以上の安全性強度を持つ n を使用することが推奨されている。例えば、もし署名者が 112 ビットの安全性強度に対応した n を使用しているとき、CA は 112 ビット以上の安全性強度に対応した n を用いることが推奨されている。SP 800-57 では、n の選択に関するさらなる議論が述べられている。この推奨ルールの例外としては、CA 間の相互認証などが考えられるが、例外としてルールを異なる運用をする場合には、さらなる分析が必要である。

ドメインパラメータの管理　ECDSA の秘密鍵と公開鍵の各ペアは、正しく特定のドメインパラメータと関連付け（例：ドメインパラメータと公開鍵が書かれた公開鍵証明書）られていることが必須とされている。ドメインパラメータは、それが失効するまでの間不正な改変から保護されることを必須としている。同じドメインパラメータを別の目的で使用しても良い。しかし、異なるベースポイント G を用いることで、鍵のペアが別の目的で、偶然正しく使用されてしまうリスクを減らすことができる。

秘密鍵と公開鍵

ECDSA の鍵ペアは、ある特定のドメインパラメータと関連付けられた、秘密鍵 d と公開鍵 Q からなる。d と Q およびドメインパラメータは、数学的に関連付けられている。秘密鍵はそれが有効なある一定期間使用される。公開鍵はそれより長く、その秘密鍵で生成された署名の検証が必要な間使用しても良い。さらなる指針については SP 800-57 に述べられている。

ECDSA の鍵ペアは、ECDSA の署名生成および署名検証にのみ用いることが必須とされている。

鍵生成アルゴリズム　秘密鍵 d と公開鍵 Q は、ドメインパラメータ

$$(q, FR, a, b, domain\_parameter\_seed, G, n, h)$$

のもとで生成される。生成方法は FIPS 186-3 Draft Appendix B.4 で与えられている（$Q = dG$）。

鍵管理

FIPS 186-3 Draft で、ECDSA での鍵管理は、DSA の鍵管理と同様に記述されている。概要については本稿 DSA の鍵管理の項を参照。

秘密情報の生成

各電子署名生成時に、その処理中で用いられる秘密かつランダムな数 k は、その処理に先立って毎回生成することが必須とされている。k の生成方法は、Appendix B.5 で与えられている。

k^{-1} は k の法 n のもとでの乗法的逆元を表す。すなわち、

$$0 < k^{-1} < n, 1 = (k^{-1}k) \mod n$$

である。この逆元 k^{-1} は、署名生成時に必要となる。k から k^{-1} を計算する方法は、Appendix C.1 で与えられている。

k^{-1} と k は、計算の際に、署名されるメッセージの情報が不要なので、あらかじめ計算しておいても良い。FIPS 186-3 Draft では、k^{-1} と k が前もって計算される場合、それらの秘匿性と完全性が保証されていることを必須としている。

署名生成アルゴリズム

FIPS 186-3 Draft において、ECDSA 署名データ (r, s) は、ANS X9.62 で規定されている方法で、上記のドメインパラメータ、秘密鍵、秘密情報、ハッシュ関数、SP 800-90 で規定された乱数生成器を用いて生成することを必須としている。

より具体的には、メッセージ M に対し、

1. ランダムな秘密情報 k を $[1, n-1]$ から選択する。
2. $r = x_1 \mod n, (x_1, y_1) = kG$ を計算する。$r = 0$ なら 1 に戻る。
3. $z = \mathrm{Hash}(M)$ の左から n のビット長分 を計算する。
4. $s = k^{-1}(z + rd) \mod n$ を計算する。

で計算された (r, s) が署名データになる。

署名検証アルゴリズム

FIPS 186-3 Draft では、署名検証も ANS X9.62 をもとに規定されている。

M', r', s' をそれぞれ M, r, s に対応する受信データであったとし、Q を署名者であると主張しているエンティティの公開鍵であるとする。このとき、署名検証処理は以下のようになる：

1. r', s' が区間 $[1, n-1]$ に入っているかを検証し、入っていなければ「INVALID」を出力して終了する。
2. $w = s'^{-1} \mod n$ を計算する。
3. $z' = \mathrm{Hash}(M')$ の左から n のビット長分 を計算する。
4. $u_1 = z'w \mod n, u_2 = r'w \mod n$ を計算する。
5. $(x_2, y_2) = u_1 G + u_2 Q$ を計算する。
6. $r' = x_1 \mod n$ であれば「VALID」を出力、そうでなければ「INVALID」を出力して終了する。

1.4.4 RSASSA-PKCS1-v1_5 と RSASSA-PSS で用いる記号と関数の定義

RSASSA-PKCS1-v1_5 および RSASSA-PSS の説明で用いる記号と記法を定義する。

d	:	RSA の署名生成鍵 (べき乗演算の指数として用いられる秘密鍵)。
d_P	:	p に関する CRT[7]の指数。$e \cdot d_P \equiv 1 \pmod{(p-1)}$ をみたす正数。
d_Q	:	q に関する CRT の署名生成鍵 (秘密鍵)。$e \cdot d_Q \equiv 1 \pmod{(q-1)}$ をみたす正数。
e	:	RSA の署名検証鍵。
EM	:	エンコードされたメッセージ (Encoded Message)。オクテット列。
$emBits$	:	EM のビット長。
$emLen$	:	EM のオクテット長。
GCD(.,.)	:	2 つの非負整数の最大公約数。
Hash	:	ハッシュ関数。
$hLen$	:	Hash の出力のオクテット長。
k	:	n のオクテット長[8]。
K	:	RSA の秘密鍵。
LCM(., $\cdots$,.)	:	複数の非負整数の最小公倍数。
m	:	文書。$0 \leq, \leq n-1$ をみたす整数。
M	:	オクテット列で表された文書。
$mask$	:	MGF の出力で得られるオクテット列。
$maskLen$	:	$mask$ のオクテット長。
MGF	:	マスク生成関数。
$mgfSeed$	:	マスクの生成に用いられる種。オクテット列。
$mLen$	:	M のオクテット長。
n	:	RSA の法[9]。$n = p \times q$。
(n, e)	:	RSA の署名検証鍵。
p, q	:	RSA の法 n の素因数。
$qInv$	:	CRT の係数。$q \cdot qInv \equiv 1 \pmod{p}$ をみたす p 未満の正数。
s	:	署名。$0 \leq s \leq n-1$ をみたす整数。
S	:	オクテット列で表された署名。
$salt$	:	RSASSA-PSS 内部で選ばれるランダムなオクテット長。
$sLen$	:	RSASSA-PSS で用いるソルト $salt$ のオクテット長。
x	:	非負整数。
X	:	オクテット列であらわされた x。
$xLen$	:	X のオクテット長。
0x	:	オクテットまたはオクテット列が 16 進表記されていることをあらわす接頭辞。“0x48” は 16 進表記のオクテット列 48 をあらわす。“(0x)48 09 0e” は、16 進表記された 3 つのオクテット 48, 09, 0e の列をあらわす。
$\oplus$	:	2 つのオクテット列のビットごとの排他的論理和。
$\lceil \cdot \rceil$	:	切り上げ関数。$\lceil x \rceil$ は実数 x 以上の最小の整数をあらわす。
$\|\|$	:	連結をあらわす演算子。
$\equiv$	:	合同記号。$a \equiv b \pmod{n}$ は、整数 n が整数 $a-b$ を割り切ることをあらわす。

整数からオクテット列への変換を行う関数とその逆関数として、I2OSP と OS2IP を以下で定義する。

[7]中国人剰余定理 (Chinise Reminder Theorem, CRT) を利用することで、合成数法のべき乗演算を高速に計算することができる。

[8]ビット長ではないことに注意。

[9]RSA PKCS#1 v2.1 では、3 つ以上の奇素数の積からなる法 n も考慮している。本章では、2 つの奇素数 p, q の積からなる法 $n = p \times q$ のみを扱う。

I2OSP$(x, xLen)$

入力 x: 変換される非負整数。

$xLen$: 出力のオクテット長。

出力 X: 長さ $xLen$ のオクテット列。

例外 “整数が大きすぎる”

手順 以下に従う。

1. $x \geq 256^{xLen}$ ならば、“整数が大きすぎる” と出力して停止する。
2. x を基数 256 で $xLen$ 桁表現する。

$$x = x_{xLen-1}256^{xLen-1} + x_{xLen-2}256^{xLen-2} + \cdots + x_1 256 + x_0$$

ここで x_i は $0 \leq x_i < 256$ をみたす整数であり、x が 256^{xLen-1} 未満の場合には 0 から始まるとする。

3. X_i を整数 x_{xLen-i} に対応するオクテットとし、$X = X_1 X_2 \ldots X_{xLen}$ を出力する $(1 \leq i \leq xLen)$。

OS2IP(X)

入力 X: オクテット列。

出力 x: 非負整数

手順

1. X を $X_1 X_2 \ldots X_{xLen}$ と表現し、x_{xLen-i} をオクテット X_i に対応する整数とする $(1 \leq i \leq xLen)$。
2. $x = x_{xLen-1}256^{xLen-1} + x_{xLen-2}256^{xLen-2} + \cdots + x_1 256 + x_0$ とする。
3. x を出力する。

RSASSA-PKCS1-v1_5 と RSASSA-PSS では、以下の鍵を用いる。k の推奨値は 256 オクテット (2048 ビット)、384 オクテット (3072 ビット) とする。

CRT (中国人剰余定理) を利用した高速化を行わない場合には、署名検証鍵は (n, e) であり、署名生成鍵は (n, d) となる。括弧書きは、鍵生成処理時にデータを秘密裏に処理するか、公開しても良いかをあらわす。

- p, q: 奇素数 (秘密)。$k/2$ オクテット。
- $n = pq$: 法 (公開)。k オクテット。
- e: 指数 (公開)。$\mathrm{GCD}(e, \mathrm{LCM}(p-1, q-1)) = 1$ をみたす。
- d: 指数 (秘密)。$e \cdot d \equiv 1 \pmod{\mathrm{LCM}(p-1, q-1)}$ をみたす。

CRT を利用した高速化により署名生成を行う場合には、署名検証鍵は (n, e) であり、署名生成鍵は $(p, q, d_P, d_Q, qInv)$ である。

- $d_P = d \mod p - 1$: 指数 (秘密)。
- $d_Q = d \mod q - 1$: 指数 (秘密)。
- $qInv$: CRT の係数。$q \cdot qInv \equiv 1 \pmod{p}$ をみたす p 未満の正数。

注意: RSA PKCS#1 v2.1 では、k として、256 オクテット (2048 ビット)、384 オクテット (3072 ビット) に加えて、128 オクテット (1024 ビット) を推奨している。電子政府の利用においては安全性の観点から、256 オクテット (2048 ビット)、384 オクテット (3072 ビット) を推奨する。

RSASSA-PKCS1-v1_5 と RSASSA-PSS は、署名生成時に文書に対してそれぞれに対応するエンコード処理を行い、得られたデータに RSASP1 を実行する (RSA 署名を生成する)。署名検証時には RSAVP1(RSA 署名の復号) を実行し、デコード処理を行う。

RSASP1(K, m)

入力 K: RSA の署名生成鍵。以下のいずれかで構成される。

- (n, d)
- $(p, q, d_P, d_Q, qInv)$

m: 文書。$0 \leq m \leq n-1$ をみたす整数。

出力 s: 署名。$0 \leq s \leq n-1$ をみたす整数。

例外 "メッセージが正当な範囲外"

前提 K が正しい RSA の署名生成鍵である。

手順 以下に従う。

1. $m \notin [0, n-1]$ ならば "メッセージが正当な範囲外" と出力して停止する。
2. s を以下で計算する。
   (a) $K = (n, d)$ ならば、$s = m^d \mod n$ とする。
   (b) $K = (p, q, d_P, d_Q, qInv)$ ならば、以下を実行する。
      i. $s_1 = m^{d_P} \mod p$, $s_2 = m^{d_Q} \mod q$ とする。
      ii. $h = (s_1 - s_2) \cdot qInv \mod p$ とする。
      iii. $s = s_2 + q \cdot h$ とする。
3. s を出力する。

RSAVP1$((n, e), s)$

入力 (n, e): RSA の署名検証鍵。

s: 署名。$0 \leq s \leq n-1$ をみたす整数。

出力 m: 文書。$0 \leq m \leq n-1$ をみたす整数。

例外 "署名が正当な範囲外"

前提 (n, e) が正しい RSA の署名検証鍵である。

手順 以下に従う。

1. $s \notin [0, n-1]$ ならば "署名が正当な範囲外" と出力して停止する。
2. $m = s^e \mod n$ とする。
3. m を出力する。

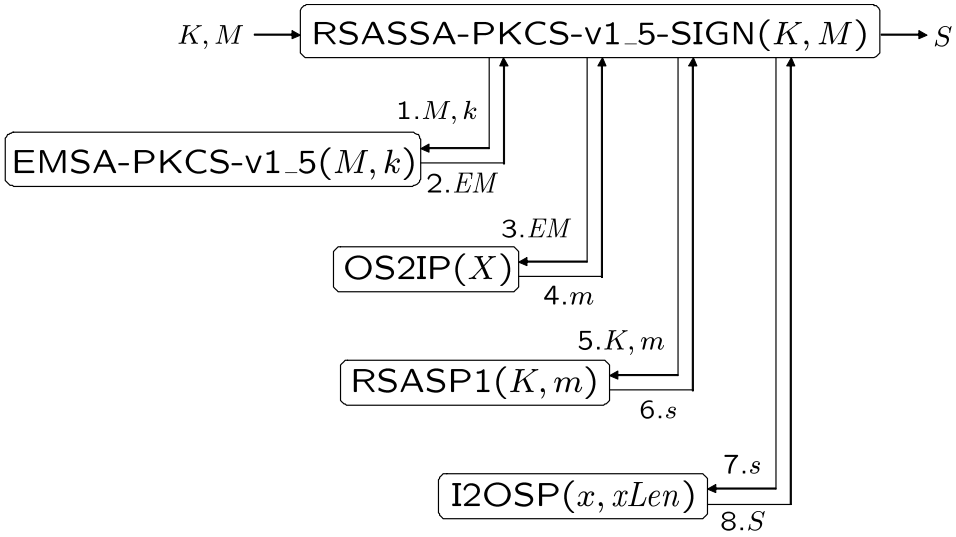

図 1: EMSA-PKCS-v1_5

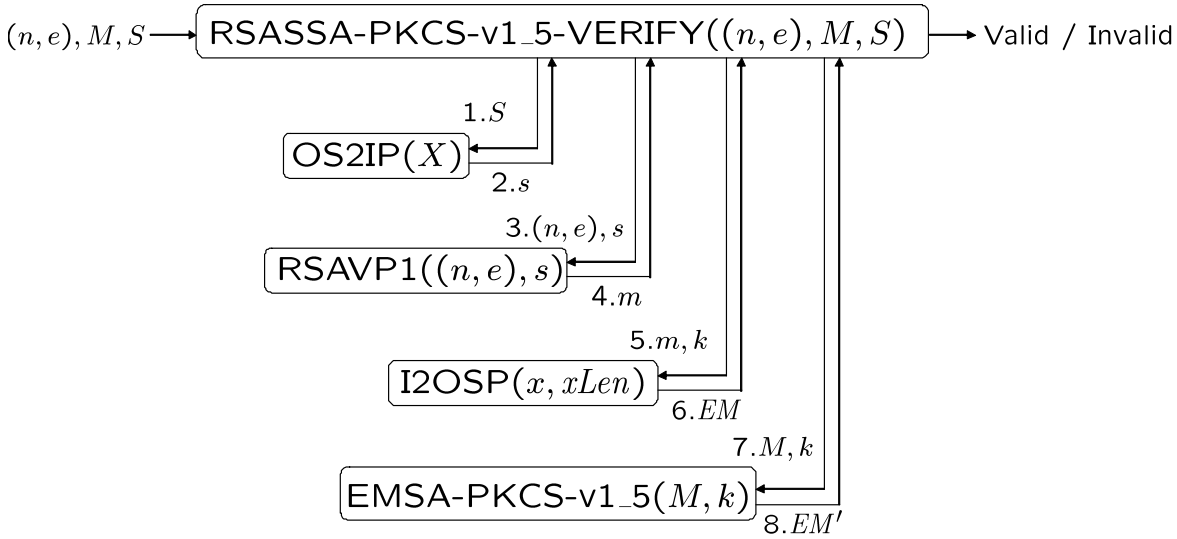

図 2: EMSA-PKCS-v1_5

1.4.5 RSASSA-PKCS1-v1_5

図 1 および図 2 に、RSASSA-PKCS1-v1_5-SIGN および、RSASSA-PKCS1-v1_5-VERIFY で用いる関数の関係を示す。

署名生成 RSASSA-PKCS1-v1_5-SIGN では、EMSA-PKCS-v1_5 を用いて文書をエンコードし、RSASP1 で署名を生成する。署名検証 RSASSA-PKCS1-v1_5-VERIFY では、EMSA-PKCS-v1_5 を用いて文書を再エンコードした結果と、署名を RSAVP1 に入力して得られる結果を比較する。まず、EMSA-PKCS-v1_5 について説明したのち、RSASSA-PKCS1-v1_5-SIGN と RSASSA-PKCS1-v1_5-VERIFY について説明する。補助とは補助情報をさす。

EMSA-PKCS-v1_5$(M, emLen)$

入力 M: エンコード対象の文書。

$emLen$: エンコード後の文書のオクテット長。DER エンコーディング T のオ

クテット長 $tLen$ に対して、$emLen \geq tLen + 11$ とする。

補助 Hash: ハッシュ関数。出力のオクテット長を $hLen$ とする。

出力 EM: エンコードされた文書。長さ $emLen$ のオクテット列。

例外 “メッセージが長すぎる”、“エンコード文書の長さが短すぎる”。

手順 以下に従う。

1. $H = \mathrm{Hash}(M)$ を計算する。ハッシュ関数が “メッセージが長すぎる” を出力したら、“メッセージが長すぎる” を出力して停止する。
2. ハッシュ関数のアルゴリズム ID とハッシュ値を、以下に示す ANS.1 の `DigestInfo` 型に DER (Design Encoding Rules) でエンコードして T を計算する。

```
DigestInfo ::= SEQUENCE{
    digestAlgorithm AlgorithmIdentifier,
    digest OCTET STRING
}
```

   T のオクテット長を $tLen$ とする。代表的なハッシュ関数について、T は下に示すように設定される。
3. $emLen < tLen + 11$ ならば、“エンコード文書の長さが短すぎる” と出力して停止する。
4. 長さ $emLen - tLen - 3$ のオクテット列 PS を 0xff を並べて設定する。PS の長さは少なくとも 8 オクテット以上である。
5. $EM = 0\mathrm{x}00||0\mathrm{x}01||PS||0\mathrm{x}00||T$ とする。
6. EM を出力する。

DER エンコーディング T は以下で計算される。

```
SHA-256: (0x)30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 || H.
SHA-384: (0x)30 41 30 0d 06 09 60 86 48 01 65 03 04 02 02 05 00 04 30 || H.
SHA-512: (0x)30 51 30 0d 06 09 60 86 48 01 65 03 04 02 03 05 00 04 40 || H.
```

注意: RSA PKCS#1 v2.1 では、ハッシュ関数 Hash として MD2、MD5、SHA-1 の利用も想定して、対応する DER エンコーディングの計算方法も示されている。電子政府の利用では、安全性の観点から、MD2、MD5、SHA-1 の利用は推奨しない。

RSASSA-PKCS1-v1_5-SIGN(K, M)

入力 K: RSA の署名生成鍵。

M 署名対象の文書。オクテット列。

出力 S: 署名。長さ k のオクテット列。ただし、k は RSA の法 n のオクテット長をあらわす。

例外 “メッセージが長すぎる”、“RSA の法が短すぎる”

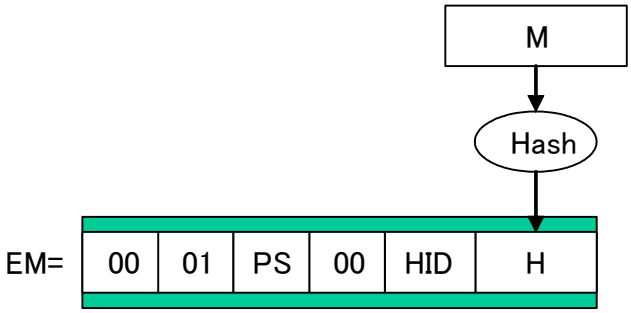

図 3: EMSA-PKCS-v1_5

手順 以下に従う。

1. 上述の EMSA-PKCS-v1_5 を利用して、長さ k のオクテット列

$$EM = \text{EMSA-PKCS-v1\_5}(M, k)$$

を求める。EM ではなく “入力メッセージが長すぎる” が出力されたら、“入力メッセージが長すぎる” を出力して停止する。“エンコード文書の長さが短すぎる” が出力されたら、“RSA の法が短すぎる” を出力して停止する。

2. RSA 署名を以下で計算する。
   (a) 1.4.4 節の OS2IP を用いて、整数

$$m = \text{OS2IP}(EM)$$

   を求める。
   (b) 1.4.4 節の RSASP1 を用いて、署名

$$s = \text{RSASP1}(K, m)$$

   を求める。
3. 1.4.4 節の I2OSP を用いて、署名

$$S = \text{I2OSP}(s, k)$$

   を求める。
4. S を出力する。

RSASSA-PKCS1-v1_5-VERIFY$((n, e), M, S)$

入力 (n, e): 署名者の RSA の署名検証鍵。

M: 検証対象の文書。オクテット列。

S: 検証対象の署名。長さ k のオクテット列。ただし、k は RSA の法 n のオクテット長をあらわす。

出力 “正当な署名” または “正当ではない署名”

例外 “入力メッセージが長すぎる”、“RSA の法が短すぎる”

手順 以下に従う。

1. 長さを検証する。オクテット列 S の長さが k でなければ、“正当ではない署名” を出力して停止する。
2. RSA の検証を以下で実行する。
   (a) 1.4.4 節の OS2IP を用いて、整数
   $$s = \text{OS2IP}(S)$$
   を求める。
   (b) 1.4.4 節の RSAVP1 を用いて、整数
   $$m = \text{RSAVP1}((n, e), s)$$
   を求める。もし、m ではなく “署名が正当な範囲外” が出力されたら、“正当ではない署名” を出力して停止する。
   (c) 1.4.4 節の I2OSP を用いて、エンコード文書
   $$EM = \text{I2OSP}(m, k)$$
   を求める。もし、EM ではなく “整数が大きすぎる” が出力されたら、“正当ではない署名” を出力して停止する。
3. 再エンコードを行う。上述の EMSA-PKCS-v1_5 を利用して、長さ k のオクテット列
   $$EM' = \text{EMSA-PKCS-v1\_5}(M, k)$$
   を求める。EM' ではなく “メッセージが長すぎる” が出力されたら、“メッセージが長すぎる” を出力して停止する。“エンコード文書の長さが短すぎる” が出力されたら、“RSA の法が短すぎる” を出力して停止する。
4. $EM = EM'$ ならば “正当な署名” を出力して停止する。そうでなければ、“正当ではない署名” を出力して停止する。

注意: Bleichenbacher の攻撃と大岩らの攻撃を防ぐために、1.3.4 節に述べたように、署名の検証時には EM の検査を適切に行う必要がある。

1.4.6 RSASSA-PSS

図 4 および図 5 に、RSASSA-PSS-SIGN および、RSASSA-PSS-VERIFY で用いる関数の関係を示す。

署名生成 RSASSA-PSS-SIGN では、EMSA-PSS-ENCODE を用いて文書をエンコードし、RSASP1 で署名を生成する。署名検証 RSASSA-PSS-VERIFY では、EMSA-PSS-VERIFY を用いて文書を再エンコードし、その過程であらわれる中間変数を比較する。まず、RSASSA-PSS-SIGN で用いる MGF について説明する。その後、EMSA-PSS-ENCODE と EMSA-PSS-VERIFY について説明したのち、RSASSA-PSS-SIGN と RSASSA-PSS-VERIFY について説明する。

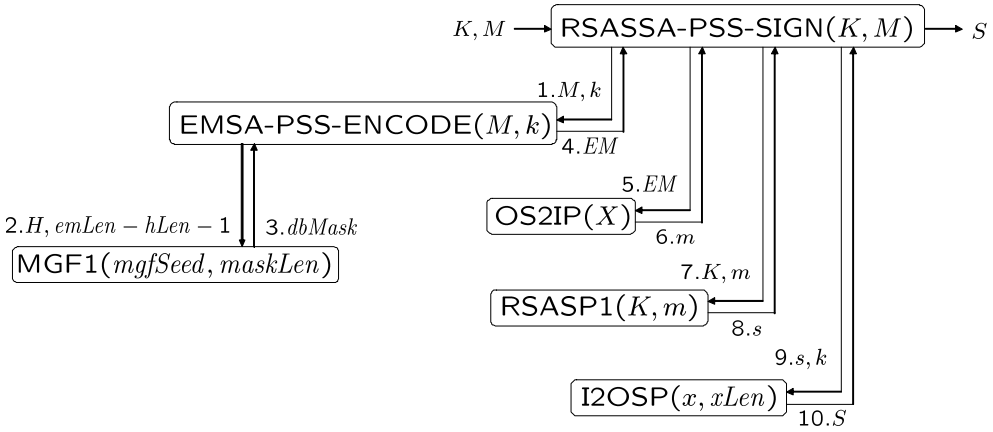

図 4: EMSA-PKCS-v1_5

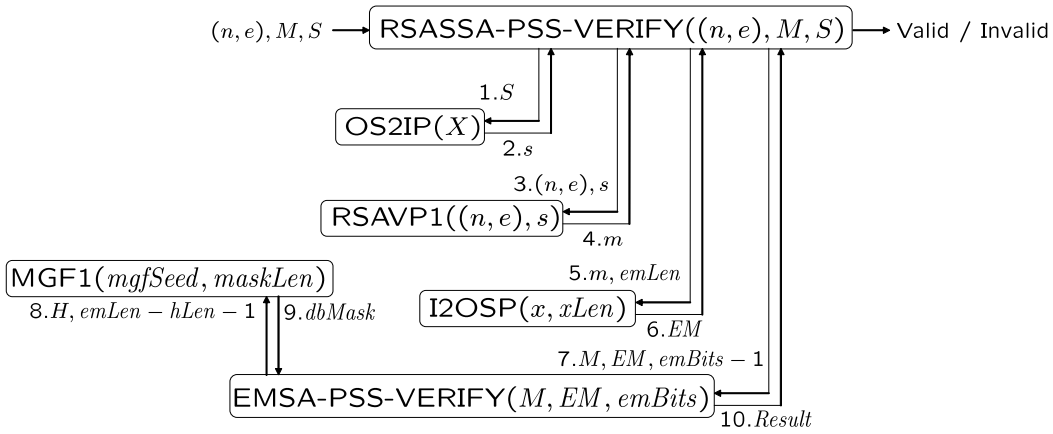

図 5: EMSA-PKCS-v1_5

MGF1(*mgfSeed*, *maskLen*)

入力 *mgfSeed*: マスクを生成するための種。オクテット列。

maskLen: 生成されるマスクのオクテット長。$maskLen \leq 2^{32} hLen$ とする。

補助 Hash: ハッシュ関数。出力のオクテット長を *hLen* とする。

出力 *mask*: マスク。長さ *maskLen* のオクテット列。

例外 “マスクが長すぎる”

手順 以下に従う。

1. $maskLen > 2^{32} hLen$ なら、“マスクが長すぎる” と出力して停止する。
2. T を空のオクテット列に設定する。
3. 以下の手順を *counter* が 0 から $\lceil maskLen/hLen \rceil$ まで実行する。
   (a) *counter* を長さ 4 のオクテット列 C に変更する。

$$C = I2OSP(counter, 4)$$

   (b) *mgfSeed* と C のハッシュ値を T に連結する。

$$T = T||\text{Hash}(mgfSeed||C)$$

4. 長さ $maskLen$ のオクテット列 T を $mask$ として出力する。

MGF1 は、オブジェクト識別子 id-mgf1 により特定される。

$$\text{id-mgf1} \quad \text{OBJECT IDENTIFIER} ::= \{\text{pkcs-1 8}\}$$

AlgorithmIdentifier の OID に関連する parameters には、MGF1 が利用するハッシュ関数を識別する hashAlgorithm を記述する。

EMSA-PSS-ENCODE$(M, emBits)$

入力 M: エンコード対象の文書。

$emBits$: エンコード後の文書の最大ビット長。$emBits \geq 8hLen + 8sLen + 9$ とする。

補助 Hash: ハッシュ関数。出力のオクテット長を $hLen$ とする。

MGF1: マスク生成関数。

$sLen$: salt のオクテット長。

出力 EM: エンコードされた文書。長さ $emLen = \lceil emBits/8 \rceil$ のオクテット列。

例外 “エンコードエラー”、“メッセージが長すぎる”

手順 以下に従う。

1. M の長さがハッシュ関数の入力の上限を超えたら、“文書が長すぎる” と出力して停止する。
2. 長さ $hLen$ のオクテット列 $mHash = \mathrm{Hash}(M)$ を計算する。
3. $emLen < hLen + sLen + 2$ なら、“エンコードエラー” と出力して停止する。
4. 長さ $sLen$ のオクテット列 $salt$ を生成する。$sLen = 0$ ならば、$salt$ は空列とする。
5. 長さ $8 + hLen + sLen$ のオクテット列

$$M' = (0\text{x})00\ 00\ 00\ 00\ 00\ 00\ 00\ 00||mHash||salt$$

を求める。上位 8 オクテットは 0x00 である。

6. $H = \mathrm{Hash}(M')$ を求める。H は長さ $hLen$ のオクテット列である。
7. 長さ $emLen - sLen - hLen - 2$ の 0x00 からなるオクテット列 PS を設定する。PS の長さは 0 でもよい。
8. 長さ $emLen - hLen - 1$ のオクテット列 $DB = PS||0\text{x}01||salt$ を設定する。
9. $dbMask = \mathrm{MGF1}(H, emLen - hLen - 1)$ を求める。
10. $maskedDB = DB \oplus dbMask$ を求める。
11. $maskedDB$ の最上位オクテットの上位 $8emLen - emBits$ ビットを 0 に設定する。
12. $EM = maskedDB||H||0\text{xbc}$ とする。
13. EM を出力する。

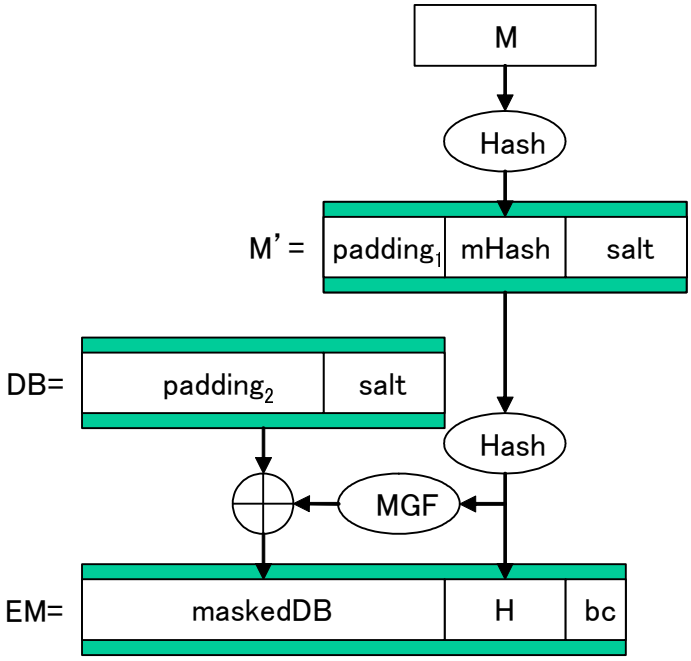

図 6: EMSA-PSS-ENCODE

EMSA-PSS-VERIFY$(M, EM, emBits)$

入力 M: 検証対象の文書。オクテット列。

EM: エンコード後の文書。長さ $emLen = \lceil emBits/8 \rceil$ のオクテット列。

$emBits$: エンコード後の文書の最大ビット長。$emBits \geq 8hLen + 8sLen + 9$ とする。

補助 Hash: ハッシュ関数。出力のオクテット長を $hLen$ とする。

MGF1: マスク生成関数。

$sLen$: salt のオクテット長。

出力 “整合”、“不整合”

手順 以下に従う。

1. M の長さがハッシュ関数の入力の上限を超えたら、“不整合” と出力して停止する。
2. 長さ $hLen$ のオクテット列 $mHash = \mathrm{Hash}(M)$ を計算する。
3. $emLen < hLen + sLen + 2$ なら、“不整合” と出力して停止する。
4. EM の最下位オクテットが 0xbc でなければ、“不整合” を出力して停止する。
5. EM の上位 $emLen - hLen - 1$ オクテットを $maskedDB$ とおき、続く $hLen$ オクテットを H とする。
6. $maskedDB$ の上位 $8emLen - emBits$ の各ビットが 0 でなければ、“不整合” を出力して停止する。

7. $dbMask = \mathrm{MGF1}(H, emLen - hLen - 1)$ を計算する。
8. $DB = maskedDB \oplus dbMask$ を計算する。
9. DB の最上位オクテットの上位 $8emLen - emBits$ ビットを 0 に設定する。
10. DB の上位 $emLen - hLen - sLen - 2$ オクテットが 0 でない、あるいは、最上位のオクテット位置を 1 とするとき 位置 $emLen - hLen - sLen - 1$ のオクテットが 0x01 ではないならば、“不整合” を出力して停止する。
11. DB の下位 $sLen$ オクテットを $salt$ とする。
12. 長さ $8 + hLen + sLen$ のオクテット列

$$M' = (0\mathrm{x})00\ 00\ 00\ 00\ 00\ 00\ 00\ 00||mHash||salt$$

を求める。上位 8 オクテットは 0x00 である。
13. $H' = \mathrm{Hash}(M')$ を求める。H は長さ $hLen$ のオクテット列である。
14. $H = H'$ ならば “整合” を出力して停止する。そうでなければ、“不整合” を出力して停止する。

RSASSA-PSS-SIGN(K, M)

入力 K: RSA の署名生成鍵。

M: 署名対象の文書。オクテット列。

出力 S: 署名。長さ k のオクテット列。ただし、k は RSA の法 n のオクテット長をあらわす。

例外 “メッセージが長すぎる”、“エンコードエラー”

手順 以下に従う。

1. 上述の EMSA-PSS-ENCODE を利用して、長さ k のオクテット列

$$EM = \mathrm{EMSA\text{-}PSS\text{-}ENCODE}(M, k)$$

を求める。EM ではなく “文書が長すぎる” が出力されたら、“文書が長すぎる” を出力して停止する。“エンコードエラー” が出力されたら、“RSA の法が短すぎる” を出力して停止する。
2. RSA 署名を以下で計算する。
   (a) 1.4.4 節の OS2IP を用いて、整数

$$m = \mathrm{OS2IP}(EM)$$

   を求める。
   (b) 1.4.4 節の RSASP1 を用いて、署名

$$s = \mathrm{RSASP1}(K, m)$$

   を求める。
3. 1.4.4 節の I2OSP を用いて、署名

$$S = \mathrm{I2OSP}(s, k)$$

   を求める。
4. S を出力する。

RSASSA-PSS-VERIFY$((n, e), M, S)$

入力 (n, e): 署名者の RSA の署名検証鍵。

M: 検証対象の文書。オクテット列。

S: 検証対象の署名。長さ k のオクテット列。ただし、k は RSA の法 n のオクテット長をあらわす。

出力 “正当な署名” または “正当ではない署名”

手順 以下に従う。

1. 長さを検証する。オクテット列 S の長さが k でなければ、“正当ではない署名” を出力して停止する。
2. RSA の検証を以下で実行する。
   (a) 1.4.4 節の OS2IP を用いて、整数

   $$s = \text{OS2IP}(S)$$

   を求める。
   (b) 1.4.4 節の RSAVP1 を用いて、整数

   $$m = \text{RSAVP1}((n, e), s)$$

   を求める。もし、m ではなく “署名が正当な範囲外” が出力されたら、“正当ではない署名” を出力して停止する。
   (c) 1.4.4 節の I2OSP を用いて、エンコード文書

   $$EM = \text{I2OSP}(m, emLen)$$

   を求める。もし、*EM* ではなく “整数が大きすぎる” が出力されたら、“正当ではない署名” を出力して停止する。
3. 上述の EMSA-PSS-VERIFY を利用して、

   $$Result = \text{EMSA-PSS-VERIFY}(M, EM, emBits - 1)$$

   を求める。
4. *Result* = 整合 ならば “正当な署名” を出力して停止する。そうでなければ、“正当ではない署名” を出力して停止する。

参考文献

[BR93] Mihir Bellare, Phillip Rogaway, “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols”, ACM Conference on Computer and Communications Security 1993, pp. 62-73, 1993

[BR96] Mihir Bellare, Phillip Rogaway, “The Exact Security of Digital Signatures - How to Sign with RSA and Rabin”, EUROCRYPT 1996, pp.399-416, 1996

[B06] Daniel Bleichenbacher, “Forging Some RSA Signatures with Pencil and Paper”, presentation in the rump session, CRYPTO 2006, 2006

[GMR88] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest, “A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks”, SIAM J. Comput. 17(2), pp.281-308, 1988

[OKW07] Yutaka Oiwa, Kazukuni Kobara, Hajime Watanabe, “New Variant for an Attack Against RSA Signature Verification Using Parameter Field”, Proceedings of EuroPKI 2007, pp.143-153, 2007.

[RSA78] Ronald L. Rivest, Adi Shamir, Leonard M. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, Commun. ACM 21(2), pp.120-126, 1978

[ANS X9.62] ANSI, “ANS X9.62 2005, Public Key Cryptography for Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)”.

[FIPS186-3] NIST, “FIPS-186-3 DRAFT Digital Signature Standard (DSS)”, `http://csrc.nist.gov/publications/PubsDrafts.html#FIPS-186--3`, 2008

[PKCS1] RSA Laboratories, “PKCS #1: RSA Cryptography Standard”, `http://www.rsa.com/rsalabs/node.asp?id=2125`

発行日　2009年5月14日　第1版　第1刷
発行者
・〒184-8795
東京都小金井市貫井北四丁目2番1号
独立行政法人 情報通信研究機構
（情報通信セキュリティ研究センター セキュリティ基盤グループ）
NATIONAL INSTITUTE OF
INFORMATION AND COMMUNICATIONS TECHNOLOGY
4-2-1 NUKUI-KITAMACHI,KOGANEI
TOKYO,184-8795 JAPAN

・〒113-6591
東京都文京区本駒込二丁目28番8号
独立行政法人 情報処理推進機構
（セキュリティセンター 暗号グループ）
INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN
2-28-8 HONKOMAGOME,BUNKYO-KU
TOKYO,113-6591 JAPAN