ANS

X9.31 1998

American National Standard for Financial Services

X9.31 -1998

Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)

Secretariat:
American Bankers Association

Approved: September 9, 1998
American National Standards Institute

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer.

Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that a concerted effort be made toward their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he has approved the standards or not from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give an interpretation of any American National Standard. Moreover, no person shall have the right or authority to issue an interpretation of an American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken to reaffirm, revise, or withdraw this standard no later than five years from the date of approval.

Published by

American Bankers Association
1120 Connecticut Ave., NW
Washington, DC 20036 USA
Customer Service Center 1(800) 338-0626 or
1(202) 663-5087
Fax 1(202) 663-7543, E-mail custserv@aba.com
X9 Online http://www.x9.org

Contents

TABLE OF FIGURES

FOREWORD

(This Foreword is included for information only and is not a part of this Standard.)

Business practice has changed with the introduction of computer-based technologies. The substitution of electronic transactions for their paper-based predecessors has reduced costs and improved efficiency. Trillions of dollars in funds and securities are transferred daily by telephone, wire services, and other electronic communications mechanisms. The high value or sheer volume of such transactions within an open environment exposes the financial community and its customers to potentially severe risks from the accidental or deliberate alteration, substitution, or destruction of data. This risk is compounded by interconnected networks and the increased number and sophistication of malicious adversaries.

Some of the conventional "due care" controls used with paper-based systems are unavailable in electronic transactions. Examples of such controls are safety paper which protects integrity, and handwritten signatures or embossed seals which indicate the intent of the originator to be legally bound. In an electronic-based environment, controls must be in place that provide the same degree of assurance and certainty as in a paper environment. The financial community is responding to these needs.

The Accredited Standards Committee on Financial Services (ANSI X9) has developed several sets of standards based on public key cryptography to protect financial information:

- X9.30-1996, *Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry* contains

  Part 1: *The Digital Signature Algorithm (DSA)* and

  Part 2: *The Secure Hash Algorithm -1 (SHA-1).*

- X9.31-1997, *Digital Signatures Using Reversible Public Key Cryptography For The Financial Services Industry (rDSA)*

- X9.62-1997, *Public Key Cryptography for the Financial Services Industry - The Elliptic Curve Digital Signature Algorithm (ECDSA)*© .

This Standard, *Digital Signatures Using Reversible Public Key Cryptography For The Financial Services Industry (rDSA)*, defines a technique for generating and validating digital signatures. When implemented with proper controls, the techniques of this standard will provide the ability to determine:

- data integrity, and
- non-repudiation of the message origin and contents.

Additionally, when used in conjunction with a Message Identifier, the techniques of this Standard provide the ability to detect duplicate transactions. It is the Committee's belief that the proper implementation of this Standard should also contribute to the enforceability of some legal obligations.

The use of this Standard, together with appropriate controls, may have considerable legal effect with respect to the apportionment of liability for erroneous or fraudulent transactions and the satisfaction of requirements for transaction enforceability. The legal implications associated with the use of this Standard may have their origin in both case law and legislation, including the Uniform Commercial Code Article 4A on Funds Transfers (Article 4A).

The details of Article 4A address (in part) the implementation of commercially reasonable security procedures and the effect of using such procedures on the apportionment of liability between a customer and a bank. A security procedure is used by Article 4A-201 "for the purpose of (i) verifying that a payment order is that of the customer, or (ii) detecting error in the transmission or the content of the payment order or communication." The commercial reasonableness of a security procedure is determined by the criteria established in Article 4A-201.

While the techniques specified in this Standard are designed to maintain the integrity of financial messages and provide the service of non-repudiation, the Standard does not guarantee that a particular implementation is secure. It is the responsibility of the financial institution to put an overall process in place with the necessary controls to ensure that the process is securely implemented. Furthermore, the controls should include the application with appropriate audit tests in order to verify compliance.

Suggestions for the improvement or revision of this standard will be welcome. They should be sent to the X9 Committee Secretariat, American Bankers Association, 1120 Connecticut Avenue, NW, Washington DC 20036.

This standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Financial Services, X9. Committee approval of the standard does not necessarily imply that all the committee members voted for its approval. At the time this standard was approved, the X9 Committee had the following members:

		Ron Bernardini
MasterCard International		Melinda Yee
		Ron Karlin
Mellon Bank, N.A.	David Taddeo	
		Genien Carlson
Merrill Lynch		John Dolan
Moore Business Forms Inc	Thomas Oswald	
National Association of Convenience Stores	Robert Swanson	
National Security Agency	Jerry Rainville	
NCR Corporation	Ms. Steve Stevens	
New York Clearing House	Vincent DeSantis	
NOVUS Services, Inc.		Thomas Kossler
		Peggy Douds
		David Pratscher
Pitney Bowes, Inc.	Leon Pintsov	
Pricewaterhousecoopers		Jeff Zimmerman
SPYRUS		Peter Yee
		Karen Randall
Unisys Corporation		Tom Hayosh
		James Graziano
VeriFone, Inc.		John Sheets
		Brad McGuinness
		Trong Nguyen
		Stuart Taylor
VISA International	Bill Chen	
Wells Fargo Bank	Tim Silva	
Xcert International, Inc.		Marc Branchaud
		Sandra Lambert

The X9F subcommittee on Data and Information Security had the following members:

Glenda Barnes, Chairman

Organization Represented	**Representative**
American Bankers Association	Kawika Daguio
American Express Company	Bonnie Howard
	Glen Weiner
Applied Communications Inc.	Cindy Rink
	Douglas Grote
Bank of America	Mack Hicks
	Kathleen Gibbons
	Richard Phillips
	Martin Johnson
Bank One Corp.	Duane Baldwin
Bankers Roundtable	Keviar Warner
	Frederick Honold
Certco LLC	Richard Ankey
	Daniel Geer
Certicom Corporation	Donald Johnson
Chase Manhattan Bank	Gene Rao
	Richard Yen
Communication Security Establishment	Alan Poplove
	Michael Chawrun
Cybersafe Corporation	Glenda Barnes
	David O'Brien
Cylink Corporation	Kamy Kavianian
	Lily Lidong Chen
Deluxe Corporation	Cory A. Surges
	Maury Jansen
	Chuck Bram
Diebold, Inc.	Sandy Morgan
	Roy Shirah
Digital Equipment Corporation	Donald Holden
Entrust Technologies	Robert Zuccherato
	Tim Moses
Ernst & Young, LLP	Richard Kastner
	Ralph Spencer Poore
Federal Reserve Bank	Richard Sweeney
	Michael Versace
	Gary Chualklin
First Data Corporation	Gene Kathol
First Union Corporation	ames Ramsey
	Sandra Lambert
Fortress Technologies	Eva Bozoki
Gilbarco, Inc.	Rena Smith
Griffin Consulting	Phillip Griffin
	Harriette Griffin
GTE Internetworking	Patrick Cain
Harmonic Systems, Inc.	Daniel Hunt
IBM Corporation	Mohammad Peyravian
	Harry Hankla
	Stephen Mike Matyas
IIT Research Institute	Roger Westman
Intel Corporation	Pam Warren
	Steve Ellis
KPMG Peat Marwick LLP	Jeffrey Stapleton
M. Blake Greenlee Associates. Ltd.	Blake Greenlee
MasterCard International	Ron Karlin

	William Poletti
Mellon Bank, N.A.	David Taddeo
Merrill Lynch	Lawrence LaBella
	John Dolan
	Ted Gerbracht
National Association of Convenience Stores	Robert Swanson
National Security Agency	Gerard Rainville
NCR	Mark Liddle
NIST	Donna Dodson
	Miles Smid
Northern Telecom, Incorporated	Warwick Ford
Northstar Technology Group, Inc.	John Bowman
Pitney Bowes, Inc.	Andrei Obrea
PricewaterhouseCoopers	John D. Hunt
	David Oshman
	Jeffrey Zimmerman
Pulse EFT Association	Karen Gardstein
	Leslie Handrix
Racal Guardata, Inc.	Scott Petersen
	Emile Soueid
	Samuel Epstein
SAIC	Wanda Gamble-Braggs
Security Dynamics	Burt Kaliski
SPYRUS	Karen Randall
	Peter Yee
Technical Communications Corporation	John Gill
TECSEC Incorporated	Edward Scheidt
	Jay Wack
US Department of Treasury	Gary Grippo
VeriFone, Inc.	John Sheets
	Stuart Taylor
	Trong Nguyen
VISA International	William Chen
Wells Fargo Bank	Azita Amini
	Terry Leahy
Xcert International	Marcus Branchaud

The X9F1 Working group which developed this standard had the following members:

Blake Greenlee, Chairman

Organization Represented	Representative
Bank One	Duane Baldwin
Booz-Allen & Hamilton Inc.	David Simonetti
Certicom	Alfred Menezes
	Don Johnson
	Scott Vanstone
	Simon Blake-Wilson
	Paul A. Lambert
CertCo	Richard Ankney
Chase Manhattan Bank	Gene Rao
Price Waterhouse Coopers	Jeff Zimmerman
	Jason Booth
CSE	Mike Chawrun
Cylink	Lily Chen
Digital Equipment Corporation	Don Holden
Griffin Consulting	Phillip Griffin
IBM	Stephen Matyas
	Alan Roginsky
Federal Reserve Bank of Atlanta	John Hannan
	Jeff Harris
Entrust	Carlyle Adams
	Robert Zuccherato
First Union	Jim Ramsay
GTE	Mort Hoffman
GTE/BBN	Pat Cain
Illinois Institute of Technology Research	R. Westman
MasterCard	Bill Poletti
M. Blake Greenlee Associates, Ltd.	Blake Greenlee
Motorola	Bob Frith
NIST	Sharon Keller
	Elaine Barker
	Miles Smid
NSA	Bob Reiter
KPMG	Jeff Stapleton
Pitney Bowes	Leon Pintsov
Lambert & Associates	Sandra Lambert
Merrill Lynch	Larry LaBella
Polaroid	Dan Bailey
IDA	Judith Freeman
Spyrus	Russ Housley

1. SCOPE

This standard, adapted from ISO/IEC 9796-2 [2] and ISO/IEC 14888-3 [16], defines a method for digital signature (signature) generation and verification for the protection of financial messages and data using reversible public key cryptography systems without message recovery. In addition, this rDSA Standard provides the criteria for the generation of public and private keys required by the algorithm and the procedural controls required for the secure use of the algorithm.

This standard guards against breaking the private key via certain factoring attacks. In particular, this standard guards against Pollard P-1 and P+1, and against difference of squares and related methods. The criterion used is that the amount of work needed for these attacks to succeed shall be at least 2^{100} arithmetic operations. The way these attacks are guarded against is by the use of strong primes, and by use of criteria between the two primes, *p* and *q*, making up the public key, where $n = pq$.

The standard guards against more modern factoring attacks such as the Elliptic Curve Method, the Quadratic Sieve, and the Number Field Sieve, by requiring that the key be sufficiently large to make these attacks infeasible.

This standard allows primes to be generated either deterministically or probabilistically where:

— A number shall accepted as prime when a probabilistic algorithm which declares it to be prime is in error with probability less than 2^{-100}.
— A deterministic prime shall be generated using a method specified in an ANSI X9 standard.
— A probabilistic prime shall be verified using primality tests specified in an ANSI X9 standard, such as X9.30-1, and as described in Appendix B: *Generation of Parameters for rDSA* of this standard.

Requirements placed upon the use of this standard, but out of scope are as follows:

— Digital signature generation and verification shall be used in conjunction with a hash algorithm specified in an ANSI X9 standard.
— Key generation shall be used in conjunctions with a random or pseudo-random number generator algorithm specified in an ANSI X9 standard.

There are various considerations to take into account for using reversible algorithms when implementing both digital signatures and encryption. Such considerations are not presented in this standard, but are provided in ANSI X9.44, *Key Transport Using Reversible Public Key Cryptography for the Financial Industry*.

Public key validation is not included in this version of the standard, but is anticipated to be added in the future.

2. DEFINITIONS, ABBREVIATIONS, AND REFERENCES

2.1 Definitions

DEFINITION	**MEANING**
Certificate (public key)	The public key and identity of an entity together with some other information, rendered unforgeable by signing the certificate with the private key of the certifying authority which issued that certificate.
Certification Authority (CA)	A Center trusted by one or more entities to create and assign certificates.
Cryptographic Hash	A (mathematical) function which maps values from a large (possibly very large) domain into a smaller range. The function satisfies the following properties: 1. it is computationally infeasible to find any input which maps to any pre-specified output; 2. it is computationally infeasible to find any two distinct inputs which map to the same output.

DEFINITION	MEANING
Cryptographic Key (Key)	A parameter that determines the operation of a cryptographic function such as: 1. the transformation from plain text to cipher text and vice versa, 2. the synchronized generation of keying material, 3. a digital signature computation or validation.
Cryptography	The discipline which embodies principles, means and methods for the transformation of data in order to hide its information content, prevent its undetected modification, prevent its unauthorized use or a combination thereof.
Cryptoperiod	The time span during which a specific key is authorized for use or in which the keys for a given system may remain in effect.
Digital Signature	The result of a cryptographic transformation of data which, when properly implemented, provides the services of: 1. origin authentication, 2. data integrity, and 3. signer non-repudiation.
Entity	A legal entity or individual, or a process or device owned or controlled by an entity or its agents.
Hash Value	The result of applying a cryptographic hash function to a message.
Key Pair	When used in public key cryptography, a public key and its corresponding private key.
Keying Material	The data (e.g., keys, certificates, and initialization vectors) necessary to establish and maintain cryptographic keying relationships.
Large Prime Factors	These are specially constructed large prime numbers, namely p_1, p_2, q_1, and q_2, each > 2^{100}, where p_1|p-1, p_2|p+1, q_1|q-1, and q_2|q+1, where *p* and *q* are the Private Prime Factors.
Legal Entity	A group or geographic area that has legal recognition, e.g., a corporation, labor union, state or nation.
m-bit number	Positive integer consisting of *m* number of bits where the high order bit, by definition, is always a "1". In the case of an *m*-bit prime number, the low order bit is also a "1" except for the 2-bit prime number “2” which has the binary value b’10’. For example, the two byte hexadecimal prime number x’01FD’ (decimal 509) is the 9-bit prime number b’0000000111111101’ represented in two bytes with 7 leading binary zeroes.
Message	The data to be signed.
Message Identifier (MID)	A field which may be used to identify a message. Typically, this field is a sequence number.
Nibble	Half a byte, i.e. 4 bits.
Non-repudiation	This service provides proof of the integrity and origin of data which can be verified by a third party.

DEFINITION	**MEANING**
Private Key	In an asymmetric (public key) cryptosystem, that key of an entity's key pair which is known only by that entity.
Private Prime Factors	The two prime numbers, namely p and q, whose product is the modulus, $pq = n$
Public Key	In an asymmetric key system, that key of an entity’s key pair which is publicly known.
Public Key Cryptography (reversible)	An asymmetric cryptographic algorithm that uses two related keys, a public key and a private key; the two keys have the property that, given the public key, it is computationally infeasible to derive the private key. Reversible public key cryptography is an asymmetric cryptographic algorithm where data encrypted using the public key can only be decrypted using the private key and conversely, data encrypted using the private key can only be decrypted using the public key.
RDSA	This standard, X9.31-1997, *Digital Signatures Using Reversible Public Key Cryptography For The Financial Services Industry*
Repudiation	The denial by an entity of having participated in part or all of a communication.
seed	Random value input into a pseudo-random number generator (PRNG) algorithm. The output of an PRNG is a random number, typically which is used as the SEED input into a key generation algorithm. Refer to Appendix A: *Random Number Generation*, where the PRNG output is hashed prior to its input to a key generation algorithm.
SEED	Random value output from either a random number generator (RNG) or a pseudo-random number generator (PRNG) used as an input value into a key generation algorithm. Refer to Appendix A: *Random Number Generation*, where the SEED is hashed prior to its input to a key generation algorithm.
Signatory	The entity that generates a digital signature on data.
Verifier	The entity that verifies the authenticity of a digital signature.

2.2 Symbols and Abbreviations

ABBREVIATION	MEANING
$\lfloor x \rfloor$	Floor function of *x*; for a given real positive *x*, $\lfloor x \rfloor = x - g$, where $\lfloor x \rfloor$ is a non-negative integer and $0 \le g < 1$.
$\lceil x \rceil$	Ceiling function of *x*; for a given real positive *x*, $\lceil x \rceil = x + g$, where $\lceil x \rceil$ is a positive integer and $0 \le g < 1$.
$\left(\frac{a}{n}\right)$	Jacobi symbol of *a* with respect to *n*.

<u>NOTE</u>
The Jacobi symbol is derived from the Legendre symbol. Let *p* be an odd prime, and let *a* be a positive integer. The Legendre symbol of the integer *a* with respect to the prime *p* may be computed by Euler's Theorem:

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p$$

The Legendre symbol of multiples of *p* with respect to the prime *p* is zero. When the integer *a* is not a multiple of a prime *p*, then the Legendre symbol of *a* with respect to *p* is valued to either +1 or -1 depending on whether *a* is or is not a square modulo *p*.

Let *n* be an odd positive integer, and let *a* be a positive integer. The Jacobi symbol of *a* with respect to *n* is the product of the Legendre symbols of *a* with respect to the prime factors of *n* (counting multiplicity of the factors).

Therefore if $n = pq$, then $\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$.

The Jacobi symbol of any integer *a* with respect to any integer *n* may be efficiently computed without the knowing the prime factors of *n* using the Law of Quadratic Reciprocity. See [6, Algorithm 2.149].

$a \vert b$	Evenly divides; e.g. *a* divides *b* evenly (with no remainder).
$\Vert$	Concatenation; e.g. A $\Vert$ B is the concatenation of A and B.
+	Addition.
$\vert x \vert$	Absolute value of *x*; $\vert x \vert$ is $-x$ if $x < 0$; otherwise it is simply *x*.
$\oplus$	Bitwise addition modulo 2 (bitwise Boolean exclusive-or)
b'01'	Binary notation used to represent one or more bits.
c	A constant
CRT	Chinese Remainder Theorem

ABBREVIATION	MEANING
d	Private (signature) exponent
e	Public (verification) exponent
GCD (a, b)	Greatest common divisor of integers a and b
H	hash function; the size of the output of H must be a multiple of 8 bits
IR	Intermediate integer
IR'	Recovered intermediate integer
k	Length of the modulus n in bits (after discarding any leading zero bits). In general, as k increases, the rDSA keys are stronger.
k_s	Length of the signature in bits, where $k_s = k - 1$
LCM (a, b)	Least common multiple of integers a and b
M	Message to be signed and sent by the signatory
M'	Message received and to be verified by the recipient. A verified digital signature shows that $M' = M$.
MID	Message Identifier
mod	Modulo
modulo n	Arithmetic modulo n
n	Modulus
p, q	Private prime factors of n
RR	Representative element of IR
s	Integer used to increase the number of bits by a fixed block size. For example, the modulus' size is 1024+256s bits where the allowed block sizes are 1024 for s=0, 1280 for s=1, 1536 for s=2, etc.
Σ	Computed Signature
Σ'	Received Signature
Sign	Signature function under the control of the private signature key
Verif	Verification function under the control of the public verification key
x'B'	Hexadecimal notation which represents one or more nibbles.
x^{-1}	Multiplicative inverse of x; for a given x and n where x and n are relatively prime, $xx^{-1} = 1 \bmod n$
$\mathbf{X_p}$, $\mathbf{X_q}$	Random numbers where the private prime factors, p and q, are the next largest primes

ABBREVIATION	MEANING
	meeting certain specified criteria, where $p > \mathbf{X_p}$ and $q > \mathbf{X_q}$
$\mathbf{X_{p1}}$, $\mathbf{X_{p2}}$, $\mathbf{X_{q1}}$, $\mathbf{X_{q2}}$	Random numbers where the large prime factors, p_1, p_2, q_1, and q_2 are the next largest primes, where $p_1 > \mathbf{X_{p1}}$, $p_2 > \mathbf{X_{p2}}$, $q_1 > \mathbf{X_{q1}}$, and $q_2 > \mathbf{X_{q2}}$, respectively.
p_1, p_2, q_1, q_2	Large prime factors of $p\pm1$ and $q\pm1$, where — p-1 has a prime factor denoted by p_1 — p+1 has a prime factor denoted by p_2 — q-1 has a prime factor denoted by q_1 — q+1 has a prime factor denoted by q_2

NOTE

1. All integers (and all strings of bits or bytes) are written with the most significant digit (or bit or byte) in the left position.
2. Those variables which are used internally during signature generation and verification, e.g., z (the length of the hash in bytes), are not included in this list of abbreviations.
3. Numbers represented by powers of 2, e.g. 2^x, are x+1 bit numbers.

2.3 References

[1] R. Rivest, A. Shamir and L. Adleman, "*A Method for Obtaining Digital Signatures and Public Key Cryptosystems*", *Communications of the ACM,* 21(2): 120-126, February, 1978.

[2] ISO/IEC, "ISO/IEC 9796-2: Digital Signature Scheme Giving Partial Message Recovery - Mechanisms Using a Hash-Function", July, 1991.

[3] L. C. Guillou, J. J. Quisquater, J.-J., M. Walker, P. Landrock, and C. Shaer, "*Precautions Taken Against Various Potential Attacks in ISO/IEC DIS 9796*", Eurocrypt '90, pp.465-473.

[4] M.O. Rabin, "*Digital Signatures and Public-Key Functions as Intractable as Factorization*", MIT Laboratory for Computer Science, Technical Report, MIT/LCS/TR-212, Jan 1979.

[5] H.C. Williams, "*A Modification Of The RSA Public-Key Encryption Process*", *IEEE Transactions on Information Theory*, 1980

[6] A. Menezes, P. C. van Oorschot, and S. Vanstone, "*Handbook of Applied Cryptography*" (HAC), CRC Press, ISBN 0-8493-8523-7, 1997

[7] Francois Morain, "*Implementation of the Goldwasser-Killian-Atkin Primality Testing Algorithm*", Project ALGO, INRIA (1988)

[8] Wieb Bosma, Doctoral Dissertation University of Amsterdam, "*Primality Proving with Cyclotomy*", (1990)

[9] I. Damgard, P. Landrock, and C. Pomerance, "*Average case error estimates for the strong probable prime test*", Math. Comp. 61 (1993), 177-194

[10] S.H. Kim and C. Pomerance, "*The probability that a random probable prime is composite*", Math. Comp. 53 (1989), 721-742

[11] P.L. Montgomery & R.D. Silverman, "*An FFT Extension to the P-1 Factoring Algorithm*", Math. Comp. 54 (1990), 839-854

[12] C. Pomerance, J.L. Selfridge, and S.S. Wagstaff Jr., "*The pseudoprimes to* $25*10^9$ ", Math. Comp. 35 (1980), 1003-1026

[13] R.D. Silverman, "*Fast Generation of Random, Strong RSA Primes*", The 1998 RSA Data Security Conference Proceedings, Cryptographer's Track, Thursday 2pm Session.

[14] J. Brillhart, D.H. Lehmer, and J.L. Selfridge, "*New primality criteria and factorizations of* $2^m \pm 1$", Math. Comp. 29 (1975), 620-647

[15] R.D. Silverman & S.S. Wagstaff Jr., "*A Practical Analysis of the Elliptic Curve Factoring Algorithm*", Math. Comp. 61 (1993) 445-462

[16] ISO/IEC, "ISO/IEC 14888-3: *Digital Signature Scheme Giving Message Recovery*", draft, 1997

[17] ANSI, "ANSI X9.30-2 *Secure Hash Algorithm* (SHA)", 1996

[18] ANSI, "ANSI X9.57 *Certificate Management*", 1997

[19] FIPS, "Federal Information Processing Standard 140-1", 1994

[20] Ueli M. Maurer, "*Fast Generation of Prime Numbers and Secure Public-Key Cryptographic Parameters*", Journal of Cryptology, 8(1995), 123-155

[21] Michael Wiener, "*Cryptanalysis of Short RSA Secret Exponents*", IEEE Transaction On Information Theory, Vol. 36, No. 3, May 1990

[22] Dan Boneh, Richard DeMillo, Richard Lipton, *"On The Importance Of Checking Cryptographic Protocols For Faults*", Advances in Cryptology - EUROCRYPT '97, May 1997, 37-51

[23] Hans Riesel, "Prime Numbers and Computer Methods for Factorization", Progress in Mathematics Vol. 57, Birkhäuser, Boston-Basel-Stuttgart, 1985

[24] J. Shawe-Taylor, Generating strong primes, *Electronics Letters*, Vol. 22, No. 16, 1986, pp. 875-877.

3. APPLICATION

3.1 General

When information is transmitted from one party to another, the recipient of information may desire to know that the information has not been altered in transit. Furthermore, the recipient may wish to be certain of the originator's identity. The use of public key cryptography digital signatures can provide assurance (1) of the identity of the signer, and (2) that the received message has not been altered during transmission.

A digital signature is an electronic analog to a written signature. The digital signature may be used in proving to a third party that the information was, in fact, signed by the originator. Unlike their written counterparts, digital signatures also verify the integrity of information. Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at any later time.

3.2 The Use of Digital Signatures

Public key cryptography is used by a *signatory* to generate a signature on data and by a *verifier* to verify the authenticity of the signature. Each signatory has a public and private key pair. The private key is used in the signature generation process, and the public key is used in the signature verification process. For both signature generation and verification, the data which is referred to in this standard as a message, M, is reduced by means of a hash algorithm specified in an ANSI X9 standard.

An adversary, who does not know the private key of the signatory, cannot feasibly generate the correct signature of the signatory. In other words, signatures cannot be forged. However, by using the signatory's public key, anyone can verify a correctly signed message.

The user of a public key requires assurance that the public key represents the owner of the key pair. That is, there must be a reliable binding of a user's identity and the user's public key. This binding may be accomplished by a mutually trusted party in the formulation of a public key certificate. This may be accomplished using a Certification Authority which generates a certificate in accordance with ANSI X9.57-1997, *Certificate Management*.

This Standard provides the ability to detect duplicate messages and prevent the acceptance of replayed messages when the signed message includes:

1. the identity of the intended recipient, and
2. a message identifier (MID).

The MID shall not repeat during the cryptoperiod of the underlying private/public key pair. Appendix B of ANSI X9.9-1986 *Financial Institution Message Authentication (Wholesale)* provides information on the use of unique MIDs.

Public and private keys shall be used for a single purpose. Digital signature key pairs shall not be used for encryption, and encryption key pairs shall not be used for digital signatures.

4. SIGNATURE ALGORITHM

This Section specifies:
— the key generation process in Section 0 *Key Generation*
— the signature process in Section 0 *Signature Generation*
— the verification process in Section 0 *Signature Verification*

Other computational methods for key generation, signature generation, and signature verification which give identical results may be implemented in conformance with this standard.

Appendix A: *Random Number Generation*	Normative
Appendix B: *Generation of Parameters for rDSA*	Informative
Appendix C: *Security Considerations*	Informative
Appendix D: *Digital Signature Examples*	Informative
Appendix E: *Implementation Considerations*	Informative

The Appendices provide additional information on the use of reversible public key cryptography.

4.1 Key Generation

Generating keys consists of the following processes:

1. Choosing the public verification exponent, e
2. Generating the private prime factors, p and q, and public modulus, n
3. Calculating the private signature exponent, d

Refer to Appendix A: *Random Number Generation* for specific algorithms.

Refer to Appendix B: *Generation of Parameters for rDSA* for specific methods.

Refer to Appendix C: *Security Considerations* for a discussion of related security issues.

Refer to Appendix D: *Digital Signature Examples* for an illustration of key generation.

Refer to Appendix E: *Implementation Considerations* for additional guidelines.

The inputs for key generation are:

1. k, the length of the modulus n in bits ($k = 1024 + 256s$, where s is an integer ≥ 0)

   NOTE
   In general, as k increases, the generated rDSA keys are stronger.

2. the exact bit length of e, the public verification exponent ($2 \leq e < 2^{k-160}$)

3. either a fixed value for $e > 0$, or 0 if e is to be randomly generated (refer to Section 0 *4.1.1* Public Verification Exponent for details)

4. (Optional) audit information consisting of:
   — hash algorithm identifier (the same hash shall be used for signature generation and signature verification)
   — (Optional) SEED value(s) for key generation
   — (Optional) ANSI approved random or pseudo number generator algorithm identifier

The outputs from key generation are:

1. a public verification key, consisting of:
   — e, the public verification exponent
   — n, the public modulus

2. a private signature key, consisting of at least one the following:
   — (Optional) d, the private signature exponent and n, the public modulus (the remainder of this standard assumes that the private signature key is both d and n), or
   — (Optional) p and q, the private prime factors, or
   — (Optional) SEED value(s) for generation of p and q
   — (Optional) calculation speed up values, (refer to Appendix E.1 *Fast Signature Algorithm*)

   Although each of the private signature key outputs are optional, enough information must be retained to regenerate d, the private signature exponent, for signature generation.

   The primes p and q shall be kept secret or destroyed. If a prime should happen to repeat for two different moduli, n, it is possible to discover[1] the private signature exponent, d. To protect against this occurrence, a random number generator (RNG) should be employed to generate the SEEDs. If a pseudo random number generator (PRNG) is used, the input seed should also be random to maximize the entropy of the output SEED. Refer to [19][2] for guidance concerning key generation and statistical random number generator tests. The SEEDs should have at least 160 bits of entropy.

   The SEEDs (used to generate p and q) must be kept secret or destroyed. However, the SEEDs may be retained with the private key as evidence that the primes were generated in an arbitrary manner. The seeds are considered to be any information which will allow the generation of the SEEDs (refer to Section 2.1 *Definitions* for the difference between the terms seed and SEED). This includes an indication of the method employed for random number generation and any starting values need to produce the

---

[1] D. Johnson, "*Greatest Common Divisor Attack*", Certicom, Jan 1997 <djohnson@certicom.com>
[2] See FIPS PUB 140-1 Sections 4.8.1 *Key Generation* and 4.11.1 *Power-Up Tests*. For further discussion on randomness, also see [6]

result. For example, in Section A.2.1, seed values include *l*, *b*, $XSEED_j$, and XKEY; in Appendix A.2.4, seed values include *V*, *DT*, and K.

3. (Optional) audit information, consisting of:
   — hash algorithm identifier (the same hash shall be used for signature generation and signature verification)
   — (Optional) ANSI approved random or pseudo number generator algorithm identifier
   — (Optional) prime number testing criteria (reserved for future use)

   Integrity and authenticity of the audit information, when present, shall be preserved.

   NOTE
   It may not be possible to store the SEEDs as part of the audit information as in the case with some key generators that may not export the random SEED value.

4.1.1 Public Verification Exponent

The public exponent, *e*, is used for signature verification, whereas the private exponent, *d*, is used for signature generation. Each signatory shall select a positive integer *e* as its public exponent, where $2 \leq e < 2^{k-160}$, and *k* is the length of the modulus *n* in bits.

The public verification exponent may be selected as a fixed value for specific applications or generated as a random value. If *e* is randomly generated, it shall be odd (both the high order bit and low order bit of *e* is a binary 1). Common fixed values for *e* include 2, 3, 17, and $2^{16}+1 = 65{,}537$.

Refer to the normative Appendix A: *Random Number Generation* for specific methods.

NOTE
When *e* is odd, the digital signature algorithm is commonly called RSA; refer to paper [1].

When *e* is even, the digital signature algorithm is commonly called Rabin-Williams; refer to papers [4] and [5].

4.1.2 Private Prime Factors and Public

Each signing entity shall secretly and randomly select two distinct positive primes, *p* and *q*, such that the following conditions are true:

1. Constraints on *p* and *q* relative to *e* are:
   — If *e* is odd, then *e* shall be relatively prime to both *p*-1 and *q*-1.
   — If *e* is even, then *p* shall be congruent to 3 mod 8, *q* shall be congruent to 7 mod 8, and *e* shall be relatively prime to both $\frac{(p-1)}{2}$ and $\frac{(q-1)}{2}$.

   The public verification exponent *e* is selected prior to generating the private prime factors, *p* and *q*.

2. The numbers $p \pm 1$ and $q \pm 1$ shall have large prime factors greater than 2^{100} and less than 2^{120}, such that:
   — *p*-1 has a prime factor denoted by p_1
   — *p*+1 has a prime factor denoted by p_2
   — *q*-1 has a prime factor denoted by q_1
   — *q*+1 has a prime factor denoted by q_2

   where the large prime factors, p_1, p_2, q_1, and q_2, are randomly selected from the set of prime numbers between 2^{100} and

2^{120}, and each shall pass at least 27 iterations of Miller-Rabin. Other prime selection criteria may be added, (but is not required to conform to this standard), refer to Appendix C.9 *Strong Primes and Safe Primes*.

Refer to Appendix B.2 *Miller-Rabin Probabilistic Primality Test* for the normative Miller-Rabin algorithm and Appendix E: *Implementation Considerations* for performance details of selecting the large prime factors.

3. The private prime factor *p* shall be the first discovered prime greater than a random number $\mathbf{X_p}$, where $(\sqrt{2})(2^{511+128s}) \leq \mathbf{X_p} \leq (2^{512+128s} - 1)$, and meets the criteria specified in Item 1 and Item 2 above, and likewise,

   the private prime factor *q* shall be the first discovered prime greater than a random number $\mathbf{X_q}$, where $(\sqrt{2})(2^{511+128s}) \leq \mathbf{X_q} \leq (2^{512+128s} - 1)$, and meets the criteria specified in Item 1 and Item 2 above.

   The random numbers, $\mathbf{X_p}$ and $\mathbf{X_q}$, shall be chosen using a random or pseudo-random number generator algorithm specified in an ANSI X9 standard.

4. The private prime factors, *p* and *q*, shall pass at least 8 rounds of the Miller-Rabin probabilistic primality test followed by a single round of the Lucas test, defined in the informative Appendix B: *Generation of Parameters for rDSA* to ensure that the probability of a private prime factor actually being a composite number is $< 2^{-100}$.

   Other primality tests of at least equivalent strength may be substituted or additionally applied. Refer to Appendix E: *Implementation Considerations* for details.

5. The private prime factors, *p* and *q*, shall be different by at least one of the first 100 bits, that is, $|p-q| > 2^{412+128s}$.

Refer to Section 0 *4.1.2.1 Generation of the Private Prime* Factors for a recommended generation method. Other key generation methods which meet these requirements are also allowed, but are not specified in this Standard. For further information on these requirements and the types of attacks they prevent, refer to Appendix C.3 *Strong Primes*.

4.1.2.1 Generation of the Private Prime Factors

The following is an approved method that satisfies the requirements of Section 0 *Private Prime Factors and Public* . The candidates for the private prime factors, p and q, are constructed using the large prime factors, p_1, p_2, q_1, and q_2, and the Chinese Remainder Theorem (CRT) [6][3].

First generate the large prime factors p_1, p_2, q_1, and q_2. This shall be done by generating four random numbers $\mathbf{X_{p1}}$, $\mathbf{X_{p2}}$, $\mathbf{X_{q1}}$, and $\mathbf{X_{q2}}$. The size of these random numbers shall be chosen from an interval, see Figure 1 *Random Number Interval* at least $[2^{100+\alpha}, 2^{101+\alpha}-1]$, such that $2^{100} \le 2^{100+\alpha} \le 2^{121}-1$. The random numbers, $\mathbf{X_{p1}}$, $\mathbf{X_{p2}}$, $\mathbf{X_{q1}}$, and $\mathbf{X_{q2}}$, shall be chosen using a random or pseudo-random number generator algorithm specified in an ANSI X9 standard. If a pseudo random number generator (PRNG) is used, the four random numbers should be generated from 4 separate input seeds.

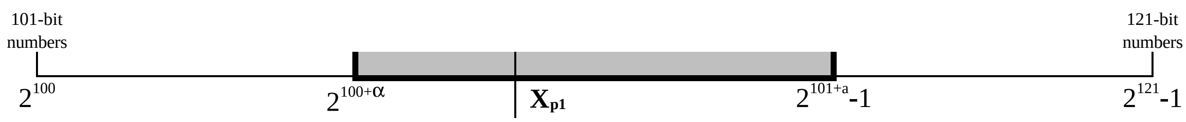


Figure 1 Random Number Interval

NOTE

The examples in Appendix D: *Digital Signature Examples* set α=0 where $\mathbf{X_{p1}}$, $\mathbf{X_{p2}}$, $\mathbf{X_{q1}}$, and $\mathbf{X_{q2}}$ are 101-bit numbers.

Then, p_1, p_2, q_1, and q_2 are the first primes greater than their respective random **X** values, and such that they are mutually prime with the public exponent e. They shall be validated with at least 27 iterations of the Miller-Rabin (or equivalent) algorithm.

To generate the private prime factor p, perform the following:

a. Generate a random number $\mathbf{X_p}$ such that $(\sqrt{2})(2^{511+128s}) \le \mathbf{X_p} \le (2^{512+128s} - 1)$
b. Compute the intermediate values:

(1) $\mathbf{R_p} = (p_2^{-1} \bmod p_1)\, p_2 - (p_1^{-1} \bmod p_2)\, p_1$

If $\mathbf{R_p} < 0$, replace $\mathbf{R_p}$ by $\mathbf{R_p} + p_1 p_2$.

(2) $\mathbf{Y}_0 = \mathbf{X_p} + (\mathbf{R_p} - \mathbf{X_p} \bmod p_1 p_2)$

If $\mathbf{Y}_0 < \mathbf{X_p}$, replace $\mathbf{Y}_0$ by $(\mathbf{Y}_0 + p_1 p_2)$.

$\mathbf{Y}_0$ is the least positive integer greater than $\mathbf{X_p}$ congruent to (1 mod p_1) and (-1 mod p_2). This means that p_1 is a large prime factor of $(\mathbf{Y}_0 - 1)$ and p_2 is a large prime factor of $(\mathbf{Y}_0 + 1)$. Search the integer sequence $\mathbf{Y}_i$ in order, where:

If e is odd, then $\mathbf{Y}_i$ is:

$\{\mathbf{Y}_0,\ \mathbf{Y}_1=\mathbf{Y}_0+(p_1 p_2),\ \mathbf{Y}_2=\mathbf{Y}_0+2(p_1 p_2),\ \mathbf{Y}_3=\mathbf{Y}_0+3(p_1 p_2),\ \ldots,\ \mathbf{Y}_j=\mathbf{Y}_0+j(p_1 p_2)\}$

If e is even, then the calculation of $\mathbf{R_p}$ in equation (1) should also add in the requirement that $\mathbf{R_p}$ = 3 mod 8 (similarly $\mathbf{R_q}$ = 7 mod 8) and $\mathbf{Y}_i$ is:

$\{\mathbf{Y}_0,\ \mathbf{Y}_1=\mathbf{Y}_0+(8p_1 p_2),\ \mathbf{Y}_2=\mathbf{Y}_0+2(8p_1 p_2),\ \mathbf{Y}_3=\mathbf{Y}_0+3(8p_1 p_2),\ \ldots,\ \mathbf{Y}_j=\mathbf{Y}_0+j(8p_1 p_2)\}$

where j is an integer ≥ 0, until:

[3] See also [6], algorithm 2.121.

— $\mathbf{Y}_i$ is prime using the primality test of Miller-Rabin with at least 8 rounds followed by a single round of the Lucas test, as defined in Appendix B: *Generation of Parameters for rDSA* and
— if e is odd, GCD ($\mathbf{Y}_i$-1, e) = 1, or
— if e is even, GCD $\left(\frac{\mathbf{Y}_i - 1}{2}, e\right) = 1$, and $\mathbf{Y}_i = 3 \bmod 8$

Then $p = \mathbf{Y}_i$. To generate the private prime factor $q = \mathbf{Y}_i$, repeat steps (a) and (b), replacing the variables $\mathbf{X_p}$ and $\mathbf{R_p}$ by $\mathbf{X_q}$ and $\mathbf{R_q}$ respectively, and if e is even, substitute $\mathbf{Y}_i = 7 \bmod 8$ for $\mathbf{Y}_i = 3 \bmod 8$ in step (b). The values $|\mathbf{X_p} - \mathbf{X_q}|$, shall be $\geq 2^{412+128s}$ and $|p - q|$ shall be $\geq 2^{412+128s}$. If not, the generation of $\mathbf{X_q}$ for finding q shall be repeated until this constraint is satisfied. If a pseudo random number generator (PRNG) is used, separate seeds should be used for $\mathbf{X_p}$ and $\mathbf{X_q}$. Refer to Appendix E.5 *Even Exponents* for further information about generating p and q using the CRT.

4.1.3 Private Signature Exponent

The private signature exponent, d, shall be a positive integer value such that $d > 2^{512+128s}$, where s is the integer $s \geq 0$, (i.e. the length of the private signature exponent must be at least half of the length of the modulus n). d is calculated as follows:

— If e is odd, then $d = e^{-1} \bmod (\mathrm{LCM}\ (p\text{-}1, q\text{-}1))$.

— If e even, then $d = e^{-1} \bmod (½\ \mathrm{LCM}\ (p\text{-}1, q\text{-}1))$.

In the extremely rare event that $d \leq 2^{512+128s}$, then the key generation process shall be repeated with new seeds for $\mathbf{X_{q1}}$, $\mathbf{X_{q2}}$, and $\mathbf{X_q}$. For further information on the size of d, refer to Appendix C.4 *Private Signature Exponent*.

4.1.4 Control of Keying Material

The signatory shall provide and maintain the proper control of all keying material. For the rDSA reversible public key cryptography digital signature algorithm, the integrity of signed data is dependent upon:

1. the prevention of unauthorized disclosure, use, modification, substitution, insertion and deletion of d, p, q, or SEEDs.
2. the prevention of unauthorized modification, substitution, insertion and deletion of e and n.

The primes p and q (the factors of the modulus n) must be kept secret or destroyed. Audit information collected during key generation may be used to recreate the key generation process, or provide evidence that a particular key pair was properly generated from specific SEEDs.

Similarly, if the private signature exponent, d, or the SEEDs are disclosed, the integrity of any message signed using that d can no longer be assured.

Public and private keys shall be used for a single purpose. Digital signature key pairs shall not be used for encryption, and encryption key pairs shall not be used for digital signatures.

NOTE
Key generation should be protected from unauthorized access to prevent disclosure of sensitive keying martial. For instance, key generation can be performed on specialized equipment or equipment that is physically isolated from normal operations, such that in the event of a hardware or software failure, no partial information is retained. For example, if a system crash causes a core dump, some of the keying material data may be captured. Using the same SEEDs will produce the same keying material that may have been compromised.

4.1.5 Public Key Validation

A key validation process for rDSA public keys is not specified in this standard. For more information, see Appendix C.8

Public Key Validation.

4.2 Signature Generation

Figure 2 Signature Generation shows the signature generation process consisting of the following steps:

1. Message Hashing,
2. Hash Encapsulation,
3. Signature Production, and
4. Signature Validation (Optional).

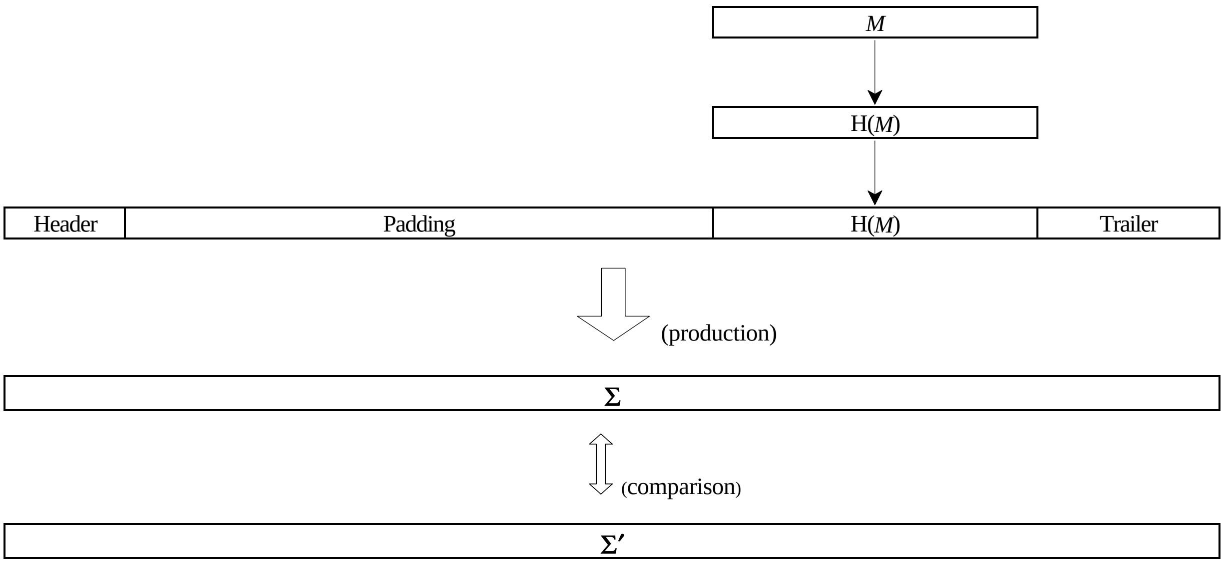


Figure 2 Signature Generation

NOTE
A good implementation of the signature process should physically protect the operations in such a way that there is no direct access to the signature function and the private signature key.

The inputs to the signature process are:
1. M, the message being signed (a bit string)
2. k, the length of the modulus n in bits ($k = 1024 + 256s$ where s is an integer ≥ 0)
3. the pair (d, n), the private signature key (refer to Section 0 *Key Generation* outputs for details of what constitutes the private signature key)
4. hash algorithm identifier
5. (Optional) audit information

Validity of the inputs should be verified prior to signature generation, including:
— the size of n is actually k bits
— the private signature key is authentic, and its integrity is verified
— the hash algorithm identifier matches the audit information (optional)

Specification of a method used to ensure the integrity of the private key is outside the scope of this Standard and is considered to be implementation dependent; however, a method to provide assurance of the integrity of the private key must exist in an implementation conforming to this standard.

The output from the signature process is:
— Σ, the digital signature (a bit string of length k_s)

1. Message Hashing

The hash function H shall be applied to the message *M*, giving the hash H(*M*). Hash algorithms used shall be specified in an ANSI X9 standard.

2. Hash Encapsulation

The hash H(*M*) shall be encapsulated within the data structure *IR* consisting of the following fields:

a. Header, 4-bit field
b. Padding, variable
c. H(*M*), variable
d. Trailer, 16-bit field

IR	Header	Padding	H(*M*)	Trailer

Lengths of ***IR*** are denoted by the following equalities:

- length (***IR***) = length (*n*)
- length (***IR***) = length (Header) + length (padding) + length (H(*M*)) + length (Trailer)

a. Header

The Header is a 4-bit field (i.e. a nibble) and shall be the hexadecimal value x'6'.

b. Padding

The padding is a variable length field, consisting of a string of nibbles with a hexadecimal value of x'B', and ending with a single nibble with a hexadecimal value of x'A'. The final nibble acts as a field separator which precedes the hash H(*M*). The length of the padding is denoted by:

length (padding) = length (*n*) - length (Header) - length (H(*M*)) - length (Trailer).

For example, the hash function SHA-1 [17] produces a 160 bit hash which requires 844+256*s* bits of padding, consisting of 210+64*s* nibbles of the hexadecimal value x'B' and a nibble of the hexadecimal value x'A', followed by the hash H(*M*).

c. Trailer

The Trailer is a fixed length field of two bytes which is constructed as follows:

— the leftmost bit shall be the binary value b'0'
— the next three bits shall encode the part number of ISO/IEC 10118 that defines the hash algorithm. For example, the part number for SHA-1 is the binary value b'011'
— the next nibble shall encode the hash algorithm as defined in the specified part number of ISO/IEC 10118. For example, the hash number for SHA-1 is the binary value b'0011'
— the last byte shall have the hexadecimal value x'CC'

For the hash function SHA-1, the Trailer shall have the hexadecimal value x'33CC'.

4. Signature Production

The signature Σ is obtained as a string of k_s bits by applying the signature function to *IR* (see step 2) under the control of the private signature key.

$$\Sigma = \textbf{Sign}(\ IR\ )$$

The signature function **Sign** is defined as follows:

The intermediate integer *IR* is a string of *k* bits computed as described above.

The representative element of *IR* with respect to *n* is denoted by *RR*.

— If *e* is odd, then *RR* is *IR*.

— If e is even and if $\left(\frac{IR}{n}\right) = +1$, then RR is IR.

— If e is even and if $\left(\frac{IR}{n}\right) = -1$, then RR is $IR/2$.

NOTE
If e is even, then $\left(\frac{RR}{n}\right) = +1$; that is, the Jacobi symbol of RR with respect to n is forced to +1.

RR shall be raised to the power d modulo n. The signature Σ is either the result or its complement to n, whichever is smaller.

$$\Sigma = \min \{ RR^d \bmod n, n-(RR^d \bmod n) \}$$

Refer to Appendix E.1 *Fast Signature Algorithm* for an alternate signature generation method.

NOTE
The signature Σ is exactly one bit less in length than the length of the modulus n.

5. Signature Validation (Optional)

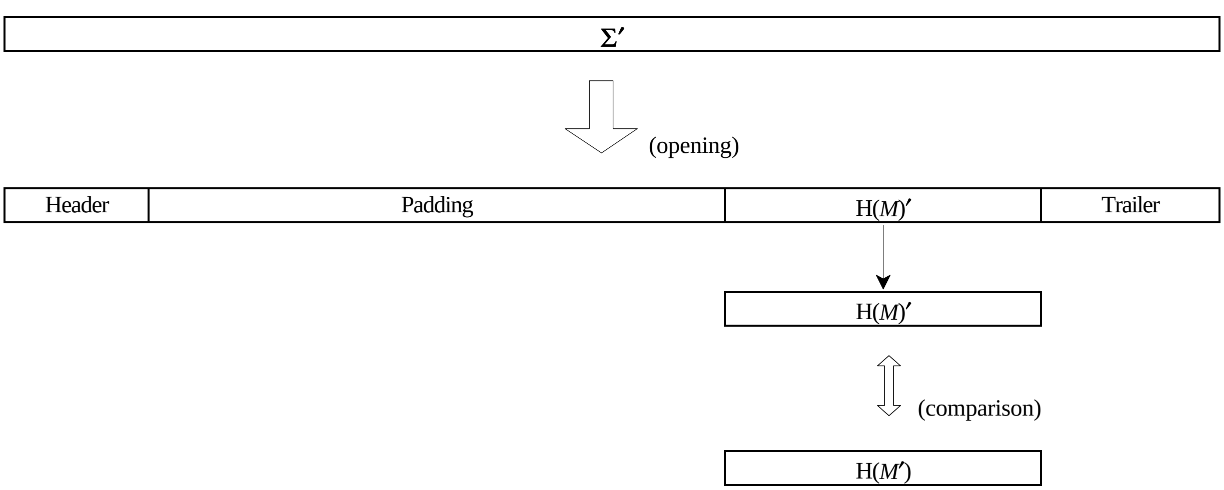
The signature should be validated by the signatory prior to sending the signed message to the receiver for verification by (1) regenerating the signature via independent processes and comparing the results, (2) verifying the signature, or (3) regenerating and comparing intermediate results during the signature process. This is a recommended procedure and is based on associated risk levels. Refer to Appendix C.5 *Cryptographic Calculation Errors*.

4.3 Signature Verification

Figure 3 Signature Verification shows that the verification process consists of the following steps:

1. Signature opening,
2. Encapsulated hash verification,
3. Hash recovery, and
4. Message hashing and comparison.

The verifier of the message shall treat the received message as M' and the signature as Σ' until the signature verification is successful, and it is proven that $M=M'$ and $\Sigma=\Sigma'$.



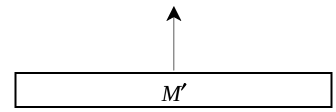


Figure 3 Signature Verification

The inputs to the verification process are:
1. Σ', the received signature (a bit string)
2. an authentic copy of the public verification key, (e, n), refer to Section 0 *Key Generation* outputs for details
3. M', the received message (a bit string)
4. hash algorithm identifier

NOTE
The method of ensuring the authenticity of the public key is outside the scope of this standard. One method to provide assurance of authenticity is by using public key certificates [18].

The validity of the inputs should be verified prior to signature verification, including:
— the size of n is actually k bits
— $2 \leq e < 2^{k-160}$
— the public key is authentic
— the hash algorithm is authentic

The output from the process is:
— an indication of signature verification success or failure.

1. Signature Opening

The signature Σ' is transformed into the recovered intermediate integer IR' by applying the verification function to Σ' under the control of the public verification key.

IR' = **Verif**(Σ')

The verification function **Verif** is defined as follows:

The signature Σ' (a positive integer less than $n/2$) shall be raised to the power e mod n in order to obtain the intermediate integer RR'. That is, $RR' = (\Sigma')^e \bmod n$.

The recovered intermediate integer IR' is then defined by the following algorithm:

If e is odd, then
— If $RR' = 12 \bmod 16$, then $IR' = RR'$;
— If $n - RR' = 12 \bmod 16$, then $IR' = n - RR'$;

If e is even and if the least significant (right most) bits of RR' are set at:

b’001’	(i.e. 1 mod 8)	then $IR' = n - RR'$
b’100’	(i.e. 4 mod 8)	then $IR' = RR'$
b’110’	(i.e. 6 mod 8)	then $IR' = 2RR'$
b’111’	(i.e. 7 mod 8)	then $IR' = 2(n - RR')$

The signature Σ' shall be rejected in all other cases, and shall also be rejected if IR' does not lie in the range from 2^{k-2} to $2^{k-1}-1$ (i.e., $2^{k-2} \leq IR' \leq 2^{k-1}-1$).

2. Encapsulated Hash Verification

The signature Σ' shall be rejected if IR' is not a string of k bits, where:

— the Header is the hexadecimal value x'6', and
— each nibble of the padding is the hexadecimal value x'B' until the final padding nibble, preceding the hash H(*M*), is the hexadecimal value x'A', and
— the rightmost byte of the Trailer field is the hexadecimal value x'CC'.

3. Hash Recovery

The Header field, the padding field, and the Trailer field shall be stripped off. The remaining bit string H(*M*)′ is the recovered hash.

4. Message Hashing and Comparison

The signature verification is accomplished by hashing the received message *M*′ and comparing it to the recovered hash, as follows:

— Compute a hash of the received message *M*′, to give a bit string H(*M*′), which is the comparative message hash.

— Verify that the hash algorithm used to compute H(*M*′) is the same as the hash algorithm specified in the Trailer field, that was used to compute the recovered hash, H(*M*)′.

— The verification process succeeds if the comparative hash H(*M*′) is the same as the recovered hash H(*M*)′, where H(*M*′) = H(*M*)′, including when *M*′ is the null message. The verifier can have a high level of confidence that the received message was sent by the party holding the private signature key.

Otherwise, if H(*M*′) ≠ H(*M*)′, then the signature shall be rejected. The message ***may*** have been modified, the message ***may*** have been incorrectly signed by the signatory, or the message ***may*** have been signed by an impostor. The message ***should*** be considered invalid.

Appendix A: Random Number Generation

(Normative)

A.1 Introduction

An implementation of this digital signature algorithm requires the ability to generate random numbers. Random numbers are used to derive a user's key pair. The random numbers are selected to be of a size which is approximately one half the length of the modulus. The numbers can be generated by either a true noise hardware randomizer or via a pseudo random function.

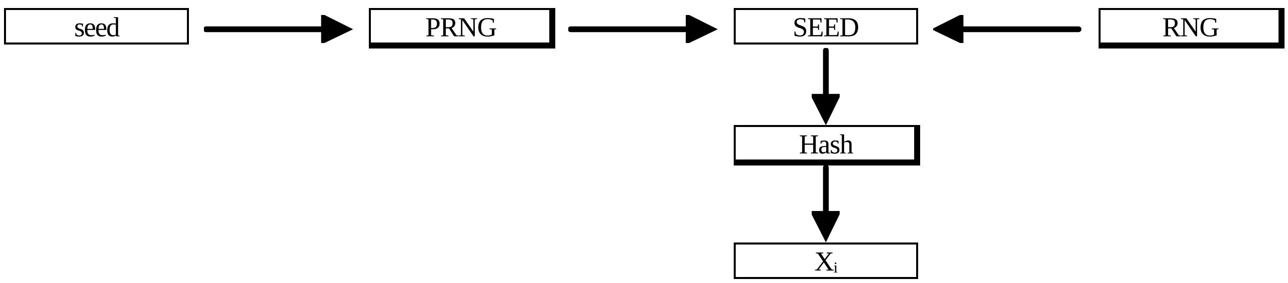


Figure A 1 Random Number Generation

Figure A 1 *Random Number Generation* shows the difference between using a random number generator (RNG) and a pseudo random number generator (PRNG). A random seed is input into the PRNG, while the SEED is the output from either the PRNG or the RNG. The SEED is then be hashed, and the resulting hash value is the random number, denoted as X_i, (if the hash value is smaller then X_i, multiple hashed SEEDs are used to produce X_i). If audit of the random number generation is not desired or not possible, (refer to Section 4.1.4 *Control of Keying Material*), then the hash is considered optional. This Appendix presents three suitable pseudo random functions as depicted in the *Random Number Generation Option Table* shown below[4].

The first two pseudo random functions employ the algorithm described in Section A.2.1 to compute a random number using a one-way function G. The algorithm employs a one-way function G(t,c) where t is 160 bits, c is b bits ($160 \leq b \leq 512$), and G(t,c) is 160 bits. Each option uses a different method of constructing G as given in Sections A.2.2 (SHA-1) and A.2.3 (DEA). In the algorithm specified in Section A.2.1, a secret b-bit key is used. If G is constructed via the SHA-1 as defined in Section A.2.2, then b is between 160 and 512 bits. If DEA is used to construct G as defined in Section A.2.3, then b is equal to 160.

Pseudo Random Functions:	**Random Number Computation**	**G based on SHA-1**	**G based on the DEA**
— **Option 1**	Section A.2.1	Section A.2.2	
— **Option 2**	Section A.2.1		Section A.2.3
— **Option 3**	Section A.2.4		

Random Number Generation Option Table

The third option is the pseudo random number function defined in [22, Appendix C] and is included in Appendix A.2.4 *Generating Pseudo Random Numbers Using the DEA* of this standard.

[4] Other ANSI X9 approved pseudo random number generation algorithms exist.

A.2 Algorithms

A.2.1 Algorithm for Computing Random Numbers

This method employs a one-way function G(t,c), where t is 160 bits in length, and c is b bits. G shall be constructed using either the Secure Hash Algorithm (SHA-1) as described in ANSI X9.30-2 (see Section A.2.2) or the Data Encryption Algorithm (DEA) describe in ANSI X3.92-81 (see Section A.2.3). If b is constructed using the SHA-1, then b shall be between 160 and 512 ($160 \leq b \leq 512$). If the DEA is used to construct G, than b is equal to 160.

Let p be the random number (SEED) to be computed, and let l be the desired length of p in bits. DIV is integer division. r is a working variable used to construct p from successive outputs of G. q is a prime number such that $2^{159} < q < 2^{160}$; it may be constructed using the algorithms defined in Appendix B: *Generation of Parameters for rDSA* (which in turn may use the algorithms presented in this Appendix).

Step 1: Compute $m = (l + 159)$ DIV 160. This is the number of iterations of G which are required. r is constructed by concatenating m outputs of the function G.

Step 2: Choose a new, secret value for the seed key, XKEY. (XKEY is of length b bits).

Step 3: In hexadecimal notation, let

t = 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0

This is the initial value for $H_0 \| H_1 \| H_2 \| H_3 \| H_4$ in the SHA-1.

Step 4: For j = 0 to m-1 do:

a. $XSEED_j$ = Optional User Input

b. $XVAL = (XKEY + XSEED_j) \bmod 2^b$

c. $x_j = G(t, XVAL) \bmod q$

d. $XKEY = (1 + XKEY + x_j) \bmod 2^b$

e. Concatenate XKEY to the current value of r

Step 5: Output l bits of r as the random number p, where $p = r \bmod 2^{l-1}$.

NOTE

XSEED and XKEY may be entered separately and under appropriate controls where dual control with split knowledge of the process of generating values of p is required.

A.2.2 Constructing the Function G from the SHA-1

G(t,c) may be constructed using steps (a)-(e) in Section 3.3 of [17]. Before executing these steps, $\{H_j\}$ and M_1 must be initialized as follows:

1. Initialize the $\{H_j\}$ by dividing the 160-bit value t into five 32-bit segments as follows:

$t = t_0 \| t_1 \| t_2 \| t_3 \| t_4$

Then $H_j = t_j$ for $j = 0$ through 4.

c = XVAL (from A.2.1)

2. There will be only one message block, M_1, which is initialized as follows:

$M_1 = c \| 0^{512-b}$

(The first b bits of M_1 contain c, and the remaining (512-b) bits are set to zero.)

3. Steps (a) through (e) of Section 3.3 of [17] are executed, and G(t,c) is the 160-bit string represented by the five words:

$H_0 \| H_1 \| H_2 \| H_3 \| H_4$

at the end of step (e).

A.2.3 Constructing the Function G from the DEA

Let $a \oplus b$ denote the bit wise exclusive-or of bit strings a and b. Suppose $a1$, $a2$, $b1$, and $b2$ are 32-bit strings. Let $b1'$ be the 24 least significant bits of $b1$. Let K = ($b1' \| b2$) and A = ($a1 \| a2$). Define:

$DEA_{(b1', b2)} (a1 \| a2) = DEA_K (A)$.

In the above, DEA_K (A) represents ordinary DEA encryption of the 64-bit block A using the 56-bit key K. Now suppose t and c are each 160 bits. To compute G(t,c):

Step 1: Write:

$t = t_1 \| t_2 \| t_3 \| t_4 \| t_5$

$c = c_1 \| c_2 \| c_3 \| c_4 \| c_5$

In the above, each t_i and c_i is 32 bits.

Step 2: For i = 1 to 5 do:

$x_i = t_i \oplus c_i$

Step 3: For i = 1 to 5 do:

$b1 = c_{((i+3) \bmod 5) + 1}$ ($b1'$ = the least significant 24 bits of $b1$)

$b2 = c_{((i+2) \bmod 5) + 1}$

$a1 = x_i$

$a2 = x_{(i \bmod 5) + 1} \oplus x_{((i+3) \bmod 5) + 1}$

$y_{i,1} \| y_{i,2} = DEA_{(b1', b2)} (a1 \| a2)$, where $y_{i,1}$ and $y_{i,2}$ = 32 bits and $b1'$ = the least significant 24 bits of $b1$.

Step 4: For i = 1 to 5 do:

$$z_i = y_{i,1} \oplus y_{((i+1) \bmod 5)+1,2} \oplus y_{((i+2) \bmod 5)+1,1}$$

Step 5: Let $G(t,c) = z_1 \| z_2 \| z_3 \| z_4 \| z_5$.

A.2.4 Generating Pseudo Random Numbers Using the DEA

Let ede*X(Y) represent the DEA multiple encryption of Y under the key *X. Let *K be a DEA key pair reserved only for the generation of pseudo random numbers, let V be a 64-bit seed value which is also kept secret, and let $\oplus$ be the exclusive-or operator. Let DT be a date/time vector which is updated on each iteration. I is an intermediate value. A 64-bit vector R is generated as follows:

I = ede*K(DT)

R = ede*K(I $\oplus$ V) and a new V is generated by V = ede*K(R $\oplus$ I).

Successive values of R may be concatenated to produce a pseudo random number of the desired length.

Appendix B: Generation of Parameters for rDSA

(Informative)

B.1 Introduction

This Appendix includes methods for generating the parameters and performing the functions needed to implement rDSA. These algorithms require a random number generator (see Appendix A: *Random Number Generation*, and an efficient modular exponentiation algorithm.

Keys generated according to this Standard have the requirement that the Pollard P $\pm$ 1 factoring attacks and repeat encryption attacks shall require in excess of 2^{100} iterations to succeed, and that the primes comprising the keys shall be generated in such a fashion that the probability that a decision procedure has incorrectly declared them to be prime is less than 2^{-100}. Refer to Appendix C: *Security Considerations* for a discussion of the relevant attacks and the work needed for their success.

The method of prime generation specified in this Appendix guarantees that for the modulus $n=pq$, $p \pm 1$ and $q \pm 1$ shall all have a prime factor of at least 101-bits. They may have even larger prime factors since they will be randomly generated.

A 1-MIPS machine performs 3.15 x 10^{13} operations in a year. For example, a Pentium Pro performs 6.3 x 10^{15} operations in a year. 2^{100} ~ 1.3 x 10^{30}. Therefore if a single iteration took just one cycle, it would take a Pentium Pro at least 2 x 10^{14} years to conduct a successful attack. Each iteration of the Pollard or repeat encryption attacks requires many operations, so the actual time will be much larger than this. The Pollard algorithms and repeat encryption attacks are inherently serial. The required operations cannot be done in parallel, so the time given above for an attack is the actual elapsed time.

The Handbook of Applied Cryptography [18, Chapter 4, Section 4.49] recommends that a probability of 2^{-80} is sufficient in practice for the probable prime tests. This standard is more stringent.

B.2 Miller-Rabin Probabilistic Primality Test

The algorithm given below may be used to generate so-called probabilistic primes. As noted in Section 0, *4.1.2.1 Generation of the Private Prime* Factors, the method presented in Section 0 satisfies the requirements of Section 0, *Private Prime Factors and Public* . Section 0 also states that other primality tests are permitted by this Standard. Refer to Appendix E.6 *Testing Candidates* for further information.

In order to generate the primes p and q, a primality test is required. There are several fast probabilistic algorithms available. The following algorithm is a simplified version of a procedure due to M.O. Rabin, based in part on ideas of Gary L. Miller and originating from an algorithm devised by John Selfridge. [See Knuth, The Art of Computer Programming, Vol. 2, Addison-Wesley, 1981, Algorithm P, page 379.] For random integers greater than 60 digits, if this algorithm is iterated c times, a false prime will be produced with probability no greater than $1/4^c$. However, for larger numbers the probability is actually lower than this. Table 4.4 of [6] gives the probability that a number of k bits which passes t iterations of Miller-Rabin is actually composite. For a 101-bit number, 27 iterations are necessary to yield a probability lower than 2^{-100}. For 512-bit primes, 8 iterations suffice to yield a probability lower than 2^{-100}. As the number gets larger, fewer iterations are necessary to ensure a probability lower than 2^{-100}. For example, for 640-bit primes, 6 iterations suffice. However, more iterations can always be applied. To test whether an odd integer is prime perform the following steps:

Step 1: Set i = 1 and c equal to the number of required iterations: 8 iterations for 512-bit prime candidate or larger, 27 iterations for prime candidates that are smaller than 512 bits. These are the minimum values; c can be increased if desired.

Step 2: Set w = the integer to be tested; set a to the largest power of 2 such that $w = 1 + 2^a m$, where m is an odd integer, and 2^a is the largest power of 2 dividing w - 1.

Step 3: Generate a random integer b in the range $1 < b < w$.

Step 4: Set j = 0 and $z = b^m \bmod w$.

Step 5: If $j = 0$ and $z = 1$, or if $z = w - 1$, go to step 9.

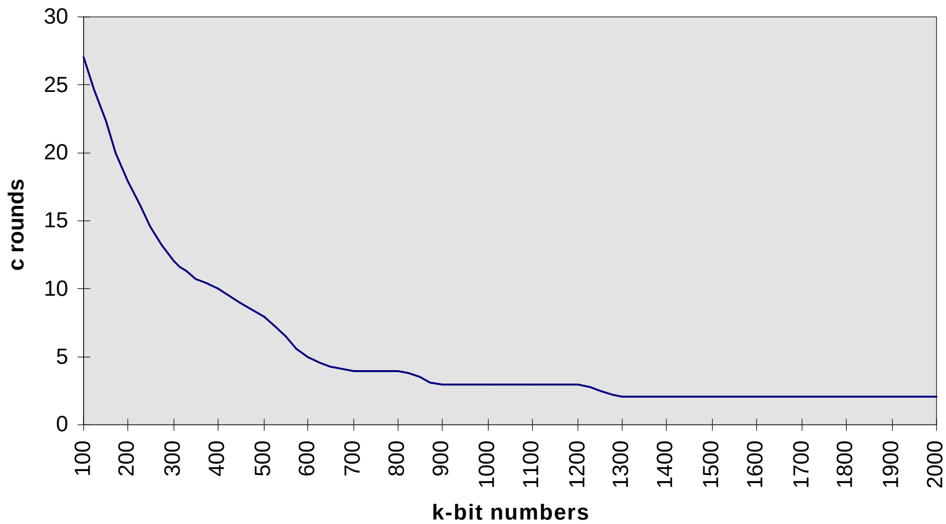
Step 6: If $j > 0$ and $z = 1$, go to step 8.

Step 7: $j = j + 1$. If $j < a$, set $z = z^2 \bmod w$ and go to step 5.

Step 8: *w* is not prime. Stop.

Step 9: If $i < c$, set $i = i + 1$ and go to step 3. Otherwise, *w* is probably prime.

The following chart shows the number of Miller-Rabin rounds, *c*, required to achieve an error probability of $< 2^{-100}$ for different *k*-bit numbers. As *k* increases, *c* decreases.



B.3 Lucas Probabilistic Primality Test

The algorithm given below is used to test whether the integer N is prime. As noted in Section, 0, *Private Prime Factors and Public* primality testing for the private prime factor, *p* and *q*, requires at least 8 rounds of the Miller-Rabin probabilistic primality test followed by a single round of the Lucas test. Refer to [23].

Find the first D in the sequence {5, -7, 9, -11, 13, -15, 17, …} for which the Jacobi symbol $\left(\frac{D}{N}\right) = -1$. Set P = 1, and $Q = \frac{1-D}{4}$. If $U_{N+1} = 0 \bmod N$, then N is probably prime. To calculate U_K where K = N + 1, from the Lucas sequence, perform the following steps:

Step 1: Let $D = (P^2 - 4Q) = (1 - 4Q)$.

Step 2: Let $K_r K_{r-1} \ldots K_0$ be the binary expansion of K.

Step 3: Set U = 1 and V= P.

Step 4: For i = r-1 to 0,

Set $(U, V) = \left(UV \bmod p, \frac{V^2 + DU^2}{2} \bmod p\right)$

If $K_i = 1$, then also set

$$(U, V) = \left(\frac{PU + V}{2} \bmod p, \frac{PV + DU}{2} \bmod p\right)$$

Step 5: $U_K = U$ and $V_K = V$

B.4 Generation of Primes

To generate a prime *p* (or *q*) of 512 + 128*s* bits, for some *s* = 0, 1, 2, 3…, perform the following steps. First *p* will be generated, followed by the generation of *q*.

Step 1: To generate *p*, set $\mathbf{X_p}$ = a random number in the range $[\sqrt{2}\,(2^{511+128s}), (2^{512+128s}) - 1]$. This shall be done using a random number generator (RNG) or pseudo random number generator (PRNG) algorithm specified in an ANSI X9 standard (see Appendix A: *Random Number Generation*), along with an appropriate seed. Likewise, to generate *q*, set $\mathbf{X_q}$ = a random number in the same range. If the absolute value of the difference $|\mathbf{X_p} - \mathbf{X_q}|$ exceeds $2^{412+128s}$, proceed to Step 2. If the absolute value does not exceed $2^{412+128s}$, keep regenerating new values of $\mathbf{X_q}$ until the difference does exceed $2^{412+128s}$. This ensures that $|p - q|$ is sufficiently large to guard against Fermat style factoring attacks, (see Appendix C: *Security Considerations* for details).

Step 2: Randomly generate four 101-bit primes, p_1, p_2, q_1, and q_2 by randomly generating four positive 101-bit integers $\mathbf{X_{p1}}$, $\mathbf{X_{p2}}$, $\mathbf{X_{q1}}$, and $\mathbf{X_{q2}}$. Sequentially search successive odd integers starting at $\mathbf{X_{p1}}$ until the first prime, p_1, is found by using A.2 to do the primality testing. Repeat this process to find p_2 starting the search at $\mathbf{X_{p2}}$, q_1 starting at $\mathbf{X_{q1}}$, and q_2 starting at $\mathbf{X_{q2}}$. Thus, p_1 is the first prime larger than $\mathbf{X_{p1}}$, p_2 is the first prime larger than $\mathbf{X_{p2}}$, q_1 is the first prime larger than $\mathbf{X_{q1}}$, and q_2 is the first prime larger than $\mathbf{X_{q2}}$.

Step 3: Apply the Chinese Remainder Theorem (twice) to compute:

$\mathbf{R_1} = (p_2^{-1} \bmod p_1)\, p_2 - (p_1^{-1} \bmod p_2)\, p_1$. Let $Y_{p0} = \mathbf{X}_p + (\mathbf{R_1} - \mathbf{X}_p \bmod p_1 p_2)$. Y_{p0} is now the first integer greater than $\mathbf{X}_p$ such that p_1 is a large prime factor of Y_{P0}-1 and p_2 is a large prime factor of Y_{p0}+1.

$\mathbf{R_2} = (q_2^{-1} \bmod q_1)\, q_2 - (q_1^{-1} \bmod q_2)\, q_1$. Let $Y_{q0} = \mathbf{X}_q + (\mathbf{R_2} - \mathbf{X}_q \bmod q_1 q_2)$. Y_{q0} is now the first integer greater than $\mathbf{X}_q$ such that q_1 is a large prime factor of Y_{q0}-1 and p_2 is a large prime factor of Y_{q0}+1.

Step 4: Check the sequence of *p* and *q* candidates such that:

If *e* is odd:

— check the sequence of *p* candidates Y_{p0}, $Y_{p0}+p_1 p_2$, $Y_{p0}+2p_1 p_2$, $Y_{p0} + 3p_1 p_2$ … to see if the public key exponent GCD(*e*, *p*-1) = 1. If so, apply the primality tests in Appendix B: *Generation of Parameters for rDSA* until the first prime is found. This shall be *p*.

— check the sequence of *q* candidates Y_{q0}, $Y_{q0}+q_1 q_2$, $Y_{q0}+2q_1 q_2$, $Y_{q0} + 3q_1 q_2$ … to see if the public key exponent GCD(*e*, q-1) = 1. If so, apply the primality tests in Appendix B: *Generation of Parameters for rDSA* until the first prime is found. This shall be *q*.

If *e* is even:

— check the sequence of *p* candidates Y_{p0}, $Y_{p0}+(8p_1 p_2)$, $Y_p+2(8p_1 p_2)$, $Y_{p0}+3(8p_1 p_2)$ … to see if the public key exponent GCD(*e*, $\frac{p-1}{2}$) = 1. If so, apply the primality tests in Appendix B: *Generation of Parameters for rDSA* until the first prime is found. This shall be *p*.

— check the sequence of *q* candidates Y_{q0}, $Y_{q0}+(8q_1 q_2)$, $Y_q+2(8q_1 q_2)$, $Y_{q0}+3(8q_1 q_2)$ … to see if the public key exponent GCD(*e*, $\frac{q-1}{2}$) = 1. If so, apply the primality tests in Appendix B: *Generation of Parameters for rDSA* until the first prime is found. This shall be q.

— it is necessary to add the additional requirements $p \equiv 3 \bmod 8$ and $q = 7 \bmod 8$. The Chinese Remainder Theorem can be used to incorporate multiple modular criteria. In step 3 above, two modular criteria are used for p, one for p_1 and one for p_2. Likewise, two modular criteria are used for q, one for q_1 and one for q_2. For example, refer to [6, Algorithm 2.121].

NOTE
For implementation details for carrying out this procedure more rapidly, refer to Appendix E: *Implementation Considerations*.

B.5 Generation of Actual Primes - Shawe-Taylor Algorithm

This section describes one method of generating actual primes. Primes generated using this method do not need to be validated using any primality tests, as they are known to be prime. This algorithm originally appeared in [24].

The following algorithm will return a prime number p containing b bits.

Random_Prime(*b*)

1. If $b < 33$ then let p be a randomly chosen prime with b bits. Since p is small, it can be randomly generated, and its primality can be proven by trial division by small primes, for example. **Return(*p*).**

2. Otherwise, do the following:

   3. If b is odd, let p_0=**Random_Prime**$((b+3)/2)$. If b is even, let p_0=**Random_Prime**$((b+2)/2)$.

   4. Choose a random integer x in the range $[2^{b-1}, 2^b]$.

   5. Let t be the least integer greater than $x/(2p_0)$.

   6. Let $y=2tp_0+1$.

   7. Pick a number a between 2 and y-2.

   8. Let $x=a^{2t} \bmod y$. If

      - x does not equal 1,
      - $\gcd(x-1,y)=1$, and
      - x raised to the power p_0 is 1 modulo y

      then p=y and **Return**(p). Otherwise, let $t=t+1$. If $2tp_0+1 > 2^b$ then let t be the least integer greater than $2^{b-1}/(2p_0)$. Return to step 6.

Appendix C: Security Considerations

(Informative)

C.1 Non-Repudiation

The non-repudiation property of the digital signature relies on the mathematical assumption that it is computationally infeasible to derive the private key from the public key and/or a set of messages and signatures prepared using the private key.

The non-repudiation property of the digital signature also relies on the practical assumption that the private key is, or can be, associated with a single entity (the signer), that only the signer has knowledge of or use of the private key, and that the private key can and will be kept secret.

The non-repudiation property would be totally undermined if the signer could easily bring a false claim that his/her private key had been compromised, and thereby be able to deny having signed messages. This potential problem is handled administratively, not technically, by means of an agreement established among the parties intending to use the digital signature protocol. The agreement should spell out the user's responsibility for protecting his/her private key, and should spell out each party's liability in the event that certain situations arise, such as the case where a user claims that his/her key has been compromised, and the user denies having signed messages.

C.2 First Party Attacks

Another potential problem is a first-party attack in which a user intentionally generates a key known to be weak, e.g., by generating primes p and q that are not strong primes. In that case, the user might be able to successfully deny having signed messages by claiming that the attacker was able to "break" the key because of the discovered weakness in the key. This potential problem must also be addressed administratively in an agreement established among the parties intending to use the digital signature protocol. This Standard addresses the issue by requiring the user to construct primes p and q, so as to ensure that p and q are strong primes.

Another way to guard against intentionally generating a weak key is to keep the random number SEEDs, if possible. A third party could re-run the prime generation algorithm for p and q to verify whether the purported weak key was generated according to the methods in this standard.

C.3 Strong Primes

Earlier drafts of this document included certain criteria which needed to be satisfied by the primes forming the modulus. These criteria are reviewed, the historical reason for each one is discussed, and why the normative prime generation procedure defined in this standard (refer to Section 0 *4.1.2.1 Generation of the Private Prime* Factors) guards against the relevant attacks. In several cases, simplified mathematical arguments are provided to show why some criteria can and should be ignored. The technical details can be found in [13].

1. If e, the public exponent is odd, then e shall be relatively prime to p-1 and q-1. This is easily satisfied by choosing e.g. e= 3, or $e=2^{16}+1$. These are commonly used values. This criterion is necessary in order for the cryptography to work properly. When constructing the primes p and q, it is easy to ensure that e does not divide LCM(p-1, q-1). If e is even, then it must be relatively prime to (p-1)/2 and (q-1)/2, and $p \neq q$ mod 8. These criteria can easily be checked by letting e be twice a prime or 2, and then generating p and q so that one of them is 3 mod 8 and the other is 7 mod 8 with e coprime to LCM(p-1,q-1). These latter conditions are easily satisfied during prime generation, refer to the procedure outlined in Appendix B.4 *Generation of Primes*.

NOTE
The public exponent e is selected prior to the generation of the primes.

2. The modulus shall have 1024+256s bits for s = 0, 1,…. As a result, the primes p and q shall then be 512+128s bits each. The choice of the value of s depends on the level of security required. Larger values of s give greater security. This

requirement is a statement reflecting the current state of the art in factoring technology. The fastest method known is the Number Field Sieve. With it, 512-bit moduli are simply not secure today for new applications, 768 bits will not be secure within just a few years, and hence a minimum of 1024 is recommended for banking. Today, one thousand fast workstations working in parallel can factor a 512-bit modulus in “about” 1-2 months. A 1024-bit modulus would be about 6 million times as difficult to factor.

3. *p* and *q* shall each pass a probabilistic test where the probability of error is less than 2^{-100}. A deterministic primality *proof* can be used, such as the Bosma-Cohen-Lenstra (a.k.a. Jacobi sum) algorithm or the Atkin-Goldwasser-Killian (a.k.a. the Elliptic Curve) algorithm can also be used [18, Sections 4.3.3 and 4.3.4]. This requirement simply states that primes shall be chosen with a high degree of confidence - that a prime is either;
   (1) chosen using a decision procedure and that the probability that the procedure is in error is less than $2^{-100} \sim 8 \times 10^{-31}$, or
   (2) that a rigorous proof of primality is used.

   It should be noted that should a composite be declared prime, then signature verification will fail except in the *extremely* unlikely case that the composite is a Carmichael number. Although exact analytic estimates for the probability have not been worked out by mathematicians, the chance of a Carmichael number passing the Miller-Rabin tests is much lower than 2^{-100}. For further details on Carmichael numbers, refer to [18, Section 4.10].

4. $p \pm 1$ and $q \pm 1$ shall each have large prime factors. In this context, 2^{100} is sufficiently large. This will put Pollard $p\pm1$ factoring attacks well out of computer range. The size of the prime factors (101 bits) of $p\pm1$ and $q\pm1$ is much too large for these algorithms to succeed in the lifetime of the universe.

5. GCD(*p*-1, *q*-1) shall be small. The method for generating primes satisfies this requirement sufficiently to guard against the repeat encryption attack. This attack requires that the order of the public exponent *e* be small in order to succeed. The argument in [15, Section 9] shows that if *r* is a large prime factor dividing LCM(*p*-1,*q*-1), then either the order of the public exponent exceeds *r* [which renders repeat encryption attacks impossible because it requires too much computation] or the order of the encrypting exponent is small with probability $< 2^{-100}$.

6. *p*/*q* shall not be near the ratio of two small integers and $|p-q| > 2^{412 + 128s}$. The purpose of these requirements is to guard against Fermat and related (i.e. Lehman) factoring algorithms. The procedure in Appendix B: *Generation of Parameters for rDSA* guarantees that |*p*-*q*| will be sufficiently large for the Fermat attack to be impossible. Similarly, the Lehman attack will require too much work to succeed as well. See the discussion in [13] for the mathematical details.

7. *p*-*q* shall have a large prime factor. This seems to be impossible to satisfy, short of actually factoring *p*-*q* or running sufficient trials of ECM to be satisfied so that no small factor exists. Furthermore, even if the condition is *not* satisfied, the relevant attack (an extension of a method of Lehmer due to R. Pinch) requires work approximately equal to $N^{1/3}$, which for N = 1024-bits, is well beyond the required minimum 2^{100} workload for this standard.

8. In earlier drafts of this standard there was also a suggestion that some forms of the modulus, such as $N = 2^{64x} \pm c$ will simplify the reduction and require less storage. This seems to have been put in place before the discovery of the Number Field Sieve (NFS). Moduli of this form are readily susceptible now to the special version of the NFS and are quite insecure. They should not be used. It would also be quite difficult to construct moduli of this form using the prime generation procedure given in Appendix B: *Generation of Parameters for rDSA.*

C.4 Private Signature Exponent

If it is known that the private signature exponent, d, is small, an attack is possible. Refer to [21] for more details. To ensure the private signature exponent is large and that the attack is not possible, this standard requires that $d > 2^{512+128s}$ where *s* is an integer ≥ 0.

C.5 Cryptographic Calculation Errors

Recently, several research groups showed that many cryptographic protocols can be broken using mistakes in calculations. In general, the results show that if the hardware or software makes certain calculation errors, then the resulting erroneous values can be used to expose secret information used by the cryptographic scheme. Refer to [22] for further details.

One such result along these lines attacks the RSA public key system. The attack is due to Boneh, Demillo, and Lipton (later improved by Lenstra) shows how a single hardware fault during the computation of an RSA digital signature completely reveals the private signature exponent. The attack applies a certain implementation of RSA known as RSA/CRT.

Multiple faults can break any RSA implementation, and a single fault can break an RSA implementation using the Chinese Remainder Theorem, as discussed in Appendix E.1 *Fast Signature Algorithm*.

In most cases, attacks on hardware faults can be defeated by checking the computed values. Consequently, inserting such tests is a recommended precaution to be implemented in security systems. Refer to Step 4. *Signature Validation* in Section 0 *Signature Generation*.

C.6 Adverse Effects on the Key Space

The strong prime requirements and limiting the size of the SEEDs to the hash function have the side effect of reducing the key space. However, this reduction presents NO reduction in cryptographic strength of the algorithms. This Appendix presents estimations of the key space to satisfy the reader's curiosity for the rationale of these techniques.

Given a number x, the number of primes < x is $\Pi(x) \approx \frac{x}{\log x}$. The factors that can reduce or affect the key space, in order of importance, are:

1. Strong prime constraints on p and q,

   For an ascending sequence $a_0, a_0+k, a_0+2k, \ldots, a_0+jk, \; j = 0, 1, 2, \ldots$, the number of primes < x where $x = a_0+jk$, for some j, is $\frac{\Pi(x)}{\Phi(k)}$, where $\Phi(k)$ is the Euler phi function. The ascending sequence is necessary to meet the strong prime constraint, (refer to Section 0, *Private Prime Factors and Public*). Recall the ascending sequence $Y_0, Y_0+p_1 p_2, \ldots, Y_0+jp_1 p_2$ in Section 0, *4.1.2.1 Generation of the Private Prime* Factors, where $k = p_1 p_2$. Hence, the number of primes < x such that (p = 1 mod p_1) and (p = -1 mod p_2) is $\frac{\Pi(x)}{\Phi(p_1 p_2)}$. Thus, the large prime factors, p_1 and p_2, are inversely proportional to the primality space, as p_1 and p_2, become large, the resulting primality space is reduced, potentially enough to create a security threat. For every bit added to p_1 and p_2, the primality space is reduced by ¼.

2. Limited size of the input SEEDs,

   When the private prime factors, p and q, are approximately 512-bits in size, there are roughly 2^{500} possible primes to choose from, which constitutes the key space. The critical factor which reduces the key space is the size of the input SEED into the pseudo-random number generator for generating the starting random numbers, $\mathbf{X_p}$ and $\mathbf{X_q}$. For example, a 100-bit SEED can only generate 2^{100} possible random numbers, which essentially reduces the key space by a factor of 2^{400}. Increasing the size of the SEED will increase the size of the potential key space. For every bit added to the SEED, the potential key space doubles in size.

3. Size of the public key verification exponent, e.

   A lesser factor that can reduce the key space is constraint 1. in Section 4.1.2 which states that when e is odd then e shall be relatively prime to both (p-1) and (q-1) and when e is even then e shall be relatively prime to both $\left(\frac{p-1}{2}\right)$ and $\left(\frac{q-1}{2}\right)$ where p = 3 mod 8 and q = 7 mod 8. This means that the value of e limits the size of the potential key space. In particular, as e becomes smaller, the key space also becomes smaller. For example, if e = 3 then approximately one third of the otherwise possible p values and one third of the possible q values are removed from consideration; hence, the key space is reduced to about $\left(1-\frac{1}{3}\right)\left(1-\frac{1}{3}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{3}\right) = \left(\frac{4}{9}\right)$ or about 44% of its otherwise potential size. For odd e, the formula for the percentage of key space size is $100\left(1-\frac{1}{e}\right)^2$. For e = 17 the key space is $\left(\frac{16}{17}\right)^2$ or about 88% of its potential size and for e = 65,537 the key space is over 99.99% of its potential size, that is, the reduction is negligible. For even e, the reduction is larger. For e = 2, the reduction is $\left(\frac{1}{2}\right)\left(\frac{1}{2}\right)\left(\frac{1}{4}\right)\left(\frac{1}{4}\right)$ which is about 1.56% of the otherwise potential key space.

Selecting e = 3 reduces the key space by 1 bit. In general, any choice of e reduces the key space by a factor of $1 - \frac{1}{e\text{-}1}$

C.7 Partial Revelation of the Private Exponent *d*

Implementation of the methods that partially reveal the private signature exponent, *d*, as explained in this Appendix present NO reduction in cryptographic strength of the algorithms, in and of themselves. This Appendix explains the partial revelation to satisfy the reader's curiosity for the rationale of these techniques. In general, some of the high order bits of *d* can always be determined for any value of *e*. Specifically:

When the public verification exponent is $e = 3$, then the private signature exponent is $d = 2(n - p - q)/3 + 1 = 2n/3 - w$, where $w = (p + q - 3)/3$. As p and q are half the size of n, this means that the top half (high order bytes) of the private exponent d can be determined. Examples in Appendices D.1 *Odd e = 3 with 1024-bit n*, D.2 *Odd e =3 with 1536-bit n*, and D.3 *Odd e =3 with 2048-bit n*, demonstrate this phenomenon. The bottom half (low order bytes) of the private signature exponent *d* cannot be determined by this calculation[5].

When $e = 2$, then $d = (n - p - q + 5)/8 = n/8 - w$ where $w = (p + q - 5)/8$. As p and q are half the size of n, this means that the top half (high order bytes) of the private exponent d can be determined. The example in Appendix *D.5 Even e =2 with 1024-bit n* demonstrates this phenomenon. The bottom half (low order bytes) cannot be determined by this calculation. For more information see [6, p.440, Algorithm 11.29: Key generation for the modified-Rabin signature scheme].

However, since $d > 2^{512+128s}$ for both cases, then at least 2^{512} bits are unknown, which satisfies the basic requirements that the probability of a successful attack $< 2^{-100}$ (in another words, the remaining key space is at least $2^{512} > 2^{100}$).

For the choice $e = 5$, then p = 2, 3, 4 mod 5, q = 2, 3, 4 mod 5, and $(p\text{-}1)(q\text{-}1)$ = 1, 2, 3, 4 mod 5. Since there are three choices for each p and q, their possible products are:

		q		
		2	**3**	**4**
p	**2**	4 = 4 mod 5	1 = 6 mod 5	3 = 8 mod 5
	3	1 = 6 mod 5	4 = 9 mod 5	2 = 12 mod 5
	4	3 = 8 mod 5	2 = 12 mod 5	1 = 16 mod 5

Hence, the probability that pq mod 5 = 1 is $\left(\frac{1}{9}\right)+\left(\frac{1}{9}\right)+\left(\frac{1}{9}\right) = \left(\frac{1}{3}\right)$, the probability that pq mod 5 = 2 is $\left(\frac{1}{9}\right)+\left(\frac{1}{9}\right) = \left(\frac{2}{9}\right)$, the probability that pq mod 5 = 3 is $\left(\frac{1}{9}\right)+\left(\frac{1}{9}\right) = \left(\frac{2}{9}\right)$, and the probability that pq mod 5 = 3 is $\left(\frac{1}{9}\right)+\left(\frac{1}{9}\right) = \left(\frac{2}{9}\right)$. Hence, an attacker guessing that $(p\text{-}1)(q\text{-}1)$ = 1 mod 5 is right with probability 1/3.

When $e = 3$, the choices for p and q are reduced, and the for $(p\text{-}1)(q\text{-}1)$ = 1 mod 3, the probability is 1. Thus, as e becomes larger, the probability that $(p\text{-}1)(q\text{-}1)$ = 1 mod e decreases, which affects the attacker's ability to select from the corresponding key space.

C.8 Public Key Validation

[5] Peter Landrock, "*How to Blackmail …*", Crypto '97 rump session at <landrock@cryptomathic.dk>

Public Key Validation is the ability for anyone using only public information to validate that a claimed public key for a particular asymmetric algorithm actually conforms to the arithmetic requirements for such a public key. Ideally, public key validation can be used to demonstrate that the existence of an associated private key is logically possible, although it (obviously) cannot determine whether such a private key actually does exist or if the claimed owner of the private key is the actual owner. A method to validate rDSA public keys is not specified in this standard. Public key validation for rDSA is a topic for research and possible standardization. This means that a user of a claimed rDSA public key has no specified way to validate the key and must trust the claim of the sender of the public key that it actually is an rDSA public key generated so as to conform to the arithmetic requirements of this standard.

C.9 Strong Primes and Safe Primes

This standard requires that a prime factor p of the modulus n be such that p-1 has a large prime factor p_1 and p+1 has a large prime factor p_2. Some cryptographers have proposed additional requirements for a prime factor of the modulus. For example, using the notation of this standard, [6, Section 4.4.2 Strong Primes] states that p is a *strong* prime if p-1 has a large prime factor p_1 (denoted r in the HAC), p+1 has a large prime factor p_2 (denoted s in the HAC) and p_1-1 (denoted r-1 in the HAC) has a large prime factor. As another example, new work[6] states that p is a *safe* prime if p-1 has a large prime factor p_1, p+1 has a large prime factor p_2, and similarly both $p_1 \pm 1$ and $p_2 \pm 1$ have a large prime factors. Safe primes are allowed by this standard, if desired, as long as all other requirements are met.

C.10 Key Size Guidelines

There are 31,536,000 seconds in a year $\approx 31 \times 10^6$, and a machine rated at one million instructions per second (1 MIPS) can therefore perform 3.1×10^{13} instructions in a year, which is the definition of a MIP Year (MY).

The Number Field Sieve (NFS) is the fastest factoring algorithm presently known. There are two phases to the algorithm; a sieving phase (see E.3 Sieving, and a linear algebra phase, both composed of numerous machine instructions. Not all instructions are created equal, however. Some execute in a single clock cycle, while others take more time. The speed with which processors execute the NFS is highly dependent on the overall machine architecture, such as cache size, memory cycle time, and pipelining efficiency. For example, a 200 MHz Pentium processor cannot execute the NFS algorithms twice as fast as a 100 MHz Pentium.

The sieving phase can be distributed among many machines. The linear algebra phase, however, requires a massive amount of memory to perform a matrix reduction. Theoretically, the matrix could be distributed among multiple machines, but at this time, no methods have been found that allow such a distributed implementation. As the size of the keys increase, not only does the run time increase, but the space requirements, composed of the memory and disk storage requirement, also increase approximately as the square root of the run time.

The function to calculate time for the NFS is $T(n) = e^{(1.91 (\log n)^{1/3} (\log \log n)^{2/3})}$, where n is the public modulus, and e is the base for the natural logarithm. Gross estimates for the relative MIPS Years, memory requirements, and disk storage requirements to factor increasing sizes of n is shown in the following chart:

Key Size (in bits)	Processing Time (MIPS Years)	Memory Requirements: Sieving Process	Memory Requirements: Linear Algebra	Storage (disk) Requirements
512 bits	4.0×10^5 MY	1.28×10^8 bytes	2.0×10^{10} bytes	5.0×10^{10} bytes
1024 bits	2.8×10^{12} MY	2.56×10^{11} bytes	1.0×10^{14} bytes	2.5×10^{14} bytes
2048 bits	2.8×10^{21} MY	8×10^{15} bytes	3.1×10^{18} bytes	7×10^{18} bytes

[6] Marc Gysin, "*Some new Pollard rho's and attacks for RSA*", Eurocrypt'97 rump session, Konstanz, Germany, May 1997 <marc@cs.uow.edu.au>

4096 bits	2.0 x 10^{33} MY	8 x 10^{21} bytes	3 x 10^{24} bytes	7 x 10^{24} bytes

Estimated Cryptographic Strength for rDSA Key Sizes

The RSA-129 challenge to factor a 129 digit public modulus (426 bits) was completed using the Quadratic Sieve in a reported 5,000 MIPS Years.

The RSA-130 challenge to factor a 130 digit public modulus (430 bits) using the Number Field Sieve was reported in a tenth of the RSA-129 time, 500 MIPS Years. However not all MIPS Years are created equal and estimates vary[7]. Using the previous definition for MIPS Years, and the NFS equation stated above, the actual processing time was closer to 16,500 MIPS Years, or 1.6 x 10^4 MIPS Years.

The estimate for a 512-bit public modulus (approximately 155 digits) should be about 25 times harder than the RSA-130 challenge, thus (1.6 x 10^4) x 25 = 4.0 x 10^5.

[7] The CrytoBytes, Vol. 2, No. 2, 1996 article and the RSA Factoring Challenge Log state that the RSA-130 challenge was completed in 500 MIPS Years, which is inconsistent with the estimates presented in this Standard as well as with the general consensus that the Number Field Sieve is about three times, not 10 times, faster than the Quadratic Sieve at the 129- to 130-digit range. This illuminates the fact that MIPS Years reports are rough estimations at best.

Appendix D: Digital Signature Examples

(Informative)

This Appendix provides five examples, four examples for an odd public verification exponent e, and one example for an even public verification exponent

D.1 Odd *e* = 3 with 1024-bit *n*

D.1.1 Key Generation

<u>Generation of *p*</u>

Parameter	Hexadecimal Value
$\mathbf{X_{p1}}$	1A1916DD B29B4EB7 EB6732E1 28
$\mathbf{X_{p2}}$	192E8AAC 41C576C8 22D93EA4 33

Note that $\mathbf{X_{p1}}$ and are $\mathbf{X_{p2}}$ 101-bit numbers, where $2^{100} \leq 2^{100+\alpha} \leq \mathbf{X_{p1}} \leq 2^{101+\alpha}-1 \leq 2^{121}-1$, $2^{100} \leq 2^{100+\alpha} \leq \mathbf{X_{p2}} \leq 2^{101+a}-1 \leq 2^{121}-1$, and $\alpha = 0$

p_1	1A1916DD B29B4EB7 EB6732E1 5B
p_2	192E8AAC 41C576C8 22D93EA4 49

Note that p_1 is the first prime number > $\mathbf{X}_{p1}$, and p_2 is the first prime > $\mathbf{X}_{p2}$.

$\mathbf{R_p}$	232C6DE7 2FCAF2EA 01B5E88F 6BCB1F7E 6BDD6618 47A83879 02D

Note that $\mathbf{R_p} = (p_2^{-1} \bmod p_1)\, p_2 - (p_1^{-1} \bmod p_2)\, p_1$

$\mathbf{X_p}$	D8CD81F0 35EC57EF E8229551 49D3BFF7 0C53520D 769D6D76 646C7A79 2E16EBD8 9FE6FC5B 605A6493 39DFC925 A86A4C6D 150B71B9 EEA02D68 885F5009 B98BD984

Note that $(\sqrt{2})(2^{511+128s}) \leq \mathbf{X_p} \leq (2^{512+128s} - 1)$, where $s = 0$

$\mathbf{Y_0}$	D8CD81F0 35EC57EF E8229551 49D3BFF7 0C53520D 769D6D76 646C7A79 2E16EBD8 9FE6FC5B 605A6704 2A3EE1D0 8331C004 7D78D253 2293AEC2 7A96978A EABEEE16

Note that $\mathbf{Y}_0 = \mathbf{X_p} + (\mathbf{R_p} - \mathbf{X_p} \bmod p_1 p_2)$

p	D8CD81F0 35EC57EF E8229551 49D3BFF7 0C53520D 769D6D76 646C7A79 2E16EBD8 9FE6FC5B 606B56F6 3EB11317 A8DCCDF2 03650EF2 8D0CB9A6 D2B2619C 52480F51

Note that $p = \mathbf{Y}_{1689} = \mathbf{Y_0} + 1689(p_1\, p_2)$, $p_1 \mid p\text{-}1$, $p_2 \mid p\text{+}1$, and GCD(e, p-1) = 1

<u>Generation of *q*</u>

Parameter Hexadecimal Value

Parameter	Hexadecimal Value				
$\mathbf{X}_{q1}$	1A5CF72E	E770DE50	CB09ACCE	A9	
$\mathbf{X}_{q2}$	134E4CAA	16D2350A	21D775C4	04	

Note that $\mathbf{X}_{q1}$ and are $\mathbf{X}_{q2}$ 101-bit numbers, where $2^{100} \leq 2^{100+\alpha} \leq \mathbf{X}_{q1} \leq 2^{101+a}-1 \leq 2^{121}-1$, $2^{100} \leq 2^{100+\alpha} \leq \mathbf{X}_{q2} \leq 2^{101+a}-1 \leq 2^{121}-1$, and $\alpha = 0$

q_1	1A5CF72E	E770DE50	CB09ACCE	B7	
q_2	134E4CAA	16D2350A	21D775C4	9F	

Note that q_1 is the first prime number > $\mathbf{X}_{q1}$, and q_2 is the first prime > $\mathbf{X}_{q2}$.

$\mathbf{R}_q$	9C72475B	057988CA	F5C0F751	88D297FB	EC85AA72
	758DE75E	28			

Note that $\mathbf{R}_q = (q_2^{-1} \bmod q_1)\, q_2 - (q_1^{-1} \bmod q_2)\, q_1$

$\mathbf{X}_q$	CC109249	5D867E64	065DEE3E	7955F2EB	C7D47A2D
	7C995338	8F97DDDC	3E1CA19C	35CA659E	DC2FC325
	6D29C262	7479C086	A699A49C	4C9CEE7E	F7BD1B34
	321DE34A				

Note that $(\sqrt{2})(2^{511+128s}) \leq \mathbf{X}_q \leq (2^{512+128s} - 1)$, where $s = 0$

$\mathbf{Y}_0$	CC109249	5D867E64	065DEE3E	7955F2EB	C7D47A2D
	7C995338	8F97DDDC	3E1CA19C	35CA659E	DC2FC2E6
	C88FE299	D52D78BE	405A97E0	1FD71DD7	819ECB91
	FA85A076				

Note that $\mathbf{Y}_0 = \mathbf{X}_q + (\mathbf{R}_q - \mathbf{X}_q \bmod q_1 q_2)$

q	CC109249	5D867E64	065DEE3E	7955F2EB	C7D47A2D
	7C995338	8F97DDDC	3E1CA19C	35CA659E	DC3D6C08
	F64068EA	FEDBD911	27F9CB7E	DC174871	1B624E30
	B857CAAD				

Note that $q = \mathbf{Y}_{1759} = \mathbf{Y}_0 + 1759(q_1 q_2)$, $q_1 \mid q-1$, $q_2 \mid q+1$, and $\mathrm{GCD}(e, q-1) = 1$

Also note that $|\mathbf{X}_p - \mathbf{X}_q| > 2^{412+128s}$, and that $|p-q| > 2^{412+128s}$, where $s = 0$

Generation of *d* and *n*

Parameter	Hexadecimal Value				
d	1CCDA20B	CFFB8D51	7EE96668	66621B11	822C7950
	D55F4BB5	BEE37989	A7D17312	E326718B	E0D79546
	EAAE87A5	6623B919	B1715FFB	D7F16028	FC400774
	1961C88C	5D7B4DAA	AC8D36A9	8C9EFBB2	6C8A4A0E
	6BC15B35	8E528A1A	C9D0F042	BEB93BCA	16B541B3
	3F80C933	A3B76928	5C462ED5	677BFE89	DF07BED5
	C127FD13	241D3C4B			

Note that $d = e^{-1} \bmod (\mathrm{LCM}\,(p-1, q-1))$

Parameter	Hexadecimal Value				
n	ACD1CC46	DFE54FE8	F9786672	664CA269	0D0AD7E5
	003BC642	7954D939	EEE8B271	52E6A947	450D7FA9
	80172DE0	64D6569A	28A83FE7	0FA840F5	E9802CB8
	984AB34B	D5C1E639	9EC21E4D	3A3A69BE	4E676F39
	5AAFEF7C	4925FD4F	AEE9F9E5	E48AF431	5DF0EC2D
	B9AD7A35	0B3DF2F4	D15DC003	9846D1AC	A3527B1A
	75049E3F	E34F43BD			

Note that $n = pq$

D.1.2 Signature Generation

Parameter	Hexadecimal Value				
M=abc	616263				
H(M)	A9993E36	4706816A	BA3E2571	7850C26C	9CD0D89D

The string x'33CC' is postfixed to the hash, indicating that the hash function is SHA-1. Padding consisting of repetitions of the nibble x'B' is prefixed to the hash. This is preceded by the Header hex value x'6'.

IR	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBAA999	3E364706	816ABA3E	25717850
	C26C9CD0	D89D33CC			

For odd e, $\boldsymbol{RR} = \boldsymbol{IR}$. The value $\boldsymbol{RR}^d \bmod n$ is now computed to be the following:

A6B496F4	A802AF90	92F1F561	931D84DB	D0B943EF
34C102B9	4DD51AB0	1E1054BC	0E0572A1	FB2DB034
569883F3	82B74E44	9F6C80C4	060FBC0F	FBD3A9CA
9D66685B	90873007	D207C1D6	4C692D01	11157BB9
76A4551E	72DDC83C	767A9D75	A4746C51	9B73CE52
C2BFBD1C	3C431D25	4FE8BB43	08FEA486	787F239F
D2944390	DA49DE45			

Since this value is greater than $n/2$, the signature $\Sigma = n - (\boldsymbol{RR}^d \bmod n)$.

Σ	61D35523	7E2A0586	6867110D	32F1D8D3	C5193F5C
	B7AC3892	B7FBE89D	0D85DB54	4E136A54	9DFCF752
	97EA9ECE	21F08558	93BBF230	99884E5E	DAC82EDF
	AE44AF04	53AB631C	CBA5C76E	DD13CBD3	D51F37FE
	40B9A5DD	64835133	86F5C704	01687DFC	27D1DDAF
	6EDBD18C	EFAD5CF8	17504C08	F482D262	AD3577AA
	2705AAF0	9056578			

D.1.3 Signature Verification

Parameter	Hexadecimal Value				
	The value $(\Sigma')^e$ mod n is computed to yield ***RR′***.				
RR′	4116108B	2429942D	3DBCAAB6	AA90E6AD	514F1C29
	44800A86	BD991D7E	332CF6B5	972AED8B	8951C3ED
	C45B7224	A91A9ADE	6CEC842B	53EC853A	2DC470FC
	DC8EF790	1A062A7D	E3066291	7E7EAE02	92ABB37D
	9EF433C0	8D6A4193	F32E3E2A	28CF3875	A2353071
	FDF1BE79	4F83495B	932778FD	16DC176E	7DE102C9
	B298016F	0AB20FF1			
	Since *n*-***RR′*** is congruent to 12 mod 16, the recovered hash ***IR′*** is set equal to *n*-***RR′***.				
IR′	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBAA999	3E364706	816ABA3E	25717850
	C26C9CD0	D89D33CC			
	Since ***IR′*** is identical to the computed hash ***IR***, and the signature verification is successful.				

D.2 Odd *e* =3 with 1536-bit *n*

D.2.1 Key Generation

Generation of *p*

Parameter	Hexadecimal Value				
X_{p1}	18272558	B6131634	8297EACA	74	
X_{p2}	1E970E8C	6C97CEF9	1F05B0FA	80	
p_1	18272558	B6131634	8297EACA	7F	
p_2	1E970E8C	6C97CEF9	1F05B0FA	93	
R_p	14D52786	AB1F0253	0F0D5A46	E660EBFB	F4FB4F0A
	C99D19F8	EF7			
X_p	F7E943C7	EF2169E9	30DCF23F	E389EF75	07EE8265
	0D42F4A0	D3A3CEFA	BE367999	BB30EE68	0B2FE064
	60F707F4	6005F8AA	7CBFCDDC	4814BBE7	F0F8BC09
	318C8E51	A48D1342	96E40D0B	BDD282DC	CBDDEE1D
	EC86F0B1	C96EAFF5	CDA70F9A	EB6EE31E	
Y_0	F7E943C7	EF2169E9	30DCF23F	E389EF75	07EE8265
	0D42F4A0	D3A3CEFA	BE367999	BB30EE68	0B2FE064
	60F707F4	6005F8AA	7CBFCDDC	4814BBE7	F0F8BC09
	318C8E51	A48D1342	96E40F4B	DB718919	27A8FBDF
	2D0A1874	86223663	3317B032	AD896B67	

Parameter	Hexadecimal Value				
p	F7E943C7	EF2169E9	30DCF23F	E389EF75	07EE8265
	0D42F4A0	D3A3CEFA	BE367999	BB30EE68	0B2FE064
	60F707F4	6005F8AA	7CBFCDDC	4814BBE7	F0F8BC09
	318C8E51	A48D1342	96E46BA6	B64B9490	FE335FC8
	397FC1F9	C06BD6C0	9ED01359	A9D30907	

Generation of *q*

Parameter	Hexadecimal Value				
X_{q1}	11FDDA6E	8128DC16	29F75192	BA	
X_{q2}	18AB178E	CA907D72	472F65E4	80	
q_1	11FDDA6E	8128DC16	29F75192	CB	
q_2	18AB178E	CA907D72	472F65E4	D3	
R_q	3C8CFDE2	135E0F05	D1931A82	AF1F6F0E	C742F81B
	9BF3E4CF	E4			
X_q	C4756001	1412D6E1	3E3E7D00	7B5C05DB	F5FF0D0F
	CFF1FA20	70D16C7A	BA93EDFB	35D87005	67E5913D
	B734E3FB	D15862EB	C59FA042	5DFA131E	549136E8
	E52397A8	ABE4705E	C4877D4F	82C4AAC6	51B33DA6
	EA14B9D5	F2A263DC	65626E4D	6CEAC767	
Y_0	C4756001	1412D6E1	3E3E7D00	7B5C05DB	F5FF0D0F
	CFF1FA20	70D16C7A	BA93EDFB	35D87005	67E5913D
	B734E3FB	D15862EB	C59FA042	5DFA131E	549136E8
	E52397A8	ABE4705E	C4877E5B	1E915AFA	811BF969
	07A58A57	B90843FB	B4068F1C	6FB006FF	
q	C4756001	1412D6E1	3E3E7D00	7B5C05DB	F5FF0D0F
	CFF1FA20	70D16C7A	BA93EDFB	35D87005	67E5913D
	B734E3FB	D15862EB	C59FA042	5DFA131E	549136E8
	E52397A8	ABE4705E	C48C3877	9AEA9679	FF196716
	68667A6C	C98AFCA9	C2D206BB	06BAEDD9	

Generation of *d* and *n*

Parameter	Hexadecimal Value				
d	7ED581A6	617C6311	465A53ED	C4155C86	807C5108
	B724070D	6C0E9935	296F4496	755CCC17	D6C15AB2
	4C6E0BB6	C2138E68	3F4746A1	B316C51E	8993DFBD
	3AC83B47	9FEAB972	B930C354	CA2DFDD3	0F2A9CB2
	22DC37B6	3B7881EE	18A7688E	0EDE30F3	8728FE7C
	8635E324	E2CD5D8E	BCAA1C51	993315FD	73B38904
	E107D7A7	B7B10EDC	A3896906	FCF87BE3	67BB858C
	A1B27E2F	C3C8674E	CC8B0F92	C0E270BA	2ECA3701
	311F68AF	CE208DCC	499B4B3D	B30FF060	5CE055D8
	93BC1461	D342EF32	E7D9720B		
n	BE404279	923A9499	E9877DE4	A6200AC9	C0BA798D
	12B60A94	2215E5CF	BE26E6E1	B00B3223	C222080B

Parameter	Hexadecimal Value				
	72A51192	231D559C	5EEAE9F2	8CA227AD	CE5DCF9B
	D82C58EB	6FE0162C	15C924FF	2F44FCBC	96BFEB0B
	344A5391	5934C2E5	24FB1CD5	164D496F	071C2183
	CC851581	C34F7B96	79E51FCB	63BA3071	0AC23C48
	9600FEF1	0C53FDDF	E6577BF7	EE8A2B77	33C53443
	23EA18DD	E80C0914	D8DF6662	66DD9C09	5CDF787C
	1A20A0A9	10A178D0	BF9F1BE7	89E4AF6F	2D36BD2B
	6790F1FD	1E8680E1	0C5421EF		

D.2.2 Signature Generation

Parameter	Hexadecimal Value				
M=abc	616263				
H(M)	A9993E36	4706816A	BA3E2571	7850C26C	9CD0D89D

The string x'33CC' is postfixed to the hash, indicating that the hash function is SHA-1. Padding consisting of repetitions of the nibble x'B' is prefixed to the hash. This is preceded by the Header hex value x'6'.

IR	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBAA999	3E364706	816ABA3E
	25717850	C26C9CD0	D89D33CC		

For odd e, $\boldsymbol{RR} = \boldsymbol{IR}$. The value $\boldsymbol{RR}^d \bmod n$ is now computed as follows:

946D0DD5	F2868BD9	2BFAA9CA	E5B4C06A	0B296A22
8D868580	83284D08	DE67E50E	18F160FD	1F65B4EC
4BB869CF	BAC320A3	2559B295	B5571764	D055B02A
33F65A7F	168773D5	7EBEEF23	23F48559	56B6420E
797498DF	A7FF067A	DBD8AC29	207F6D56	797FD610
881045BB	3C14AC9B	1E3AF3C4	DCACC8F9	5681AFDE
15A8C300	561FAFE1	B6333E07	DEC1A25C	96A94896
04ACB54D	DA7FA319	614C02B9	3D17C457	3D9F3214
257E6D5D	93948F56	D5C5D55A	EF4E4AB8	22AB91D4
97CD71A8	C2763E33	B9EDF77C		

Since this value is greater than $n/2$, the signature $\boldsymbol{\Sigma} = n - (\boldsymbol{RR}^d \bmod n)$.

Σ	29D334A3	9FB408C0	BD8CD419	C06B4A5F	B5910F6A
	852F8513	9EED98C6	DFBF01D3	9719D126	A2BC531F
	26ECA7C2	685A34F9	3991375C	D74B1048	FE081F71
	A435FE6C	5958A256	970A35DC	0B507763	4009A8FC
	BAD5BAB1	B135BC6A	492270AB	F5CDDC18	8DA04B73
	4474CFC6	873ACEFB	5BAA2C06	870D6777	B4408C6A

Parameter	Hexadecimal Value				
	80583BF0	B6344DFE	30243DF0	0FC8891A	9D1BEBAD
	1F3D6390	0D8C65FB	779363A9	29C5D7B2	1F404667
	F4A2334B	7D0CE979	E9D9468C	9A9664B7	0A7C2B56
	CFC38054	5C1042AD	52662A73		

D.2.3 Signature Verification

Parameter Hexadecimal Value

The value $(\Sigma')^e$ mod n is computed to yield ***RR′***.

Parameter	Hexadecimal Value				
RR′	528486BD	D67ED8DE	2DCBC228	EA644F0E	04FEBDD1
	56FA4ED8	665A2A14	026B2B25	F44F7668	06664C4F
	B6E955D6	676199E0	A32F2E36	D0E66BF2	12A213E0
	1C709D2F	B4245A70	5A0D6943	73894100	DB042F4F
	788E97D5	9D790729	693F6119	5A918DB3	4B6065C8
	10C959C6	0793BFDA	BE29640F	A7FE74B5	4F06808C
	DA454335	50984224	2A9BC03C	32CE6FBB	78097887
	682E5D22	2C504D49	1D23AAA6	AB21E04D	A123BCC0
	5E64E4ED	54E5BD15	03E4724E	4BAE6868	ABCC02ED
	421F79AC	5C19E410	33B6EE23		

Since n - ***RR′*** is congruent to 12 mod 16, the recovered hash ***IR′*** is set equal to n - ***RR′***. Since ***IR′*** is identical to the computed hash ***IR***, and the signature verification is successful.

D.3 Odd *e* =3 with 2048-bit *n*

D.3.1 Key Generation

<u>Generation of *p*</u>

Parameter	Hexadecimal Value				
X_{p1}	1A1916DD	B29B4EB7	EB6732E1	28	
X_{p2}	192E8AAC	41C576C8	22D93EA4	33	
p_1	1A1916DD	B29B4EB7	EB6732E1	5B	
p_2	192E8AAC	41C576C8	22D93EA4	49	
R_p	232C6DE7	2FCAF2EA	01B5E88F	6BCB1F7E	6BDD6618
	47A83879	02D			
X_p	E532CF6E	17E192B6	94E1313A	3CE04353	4CD24E26
	4E773F7B	DE8D9A19	0EF1A5AB	841C4D39	32A85B48
	D428875D	D7101764	89E6973D	CBB9DC37	9432C26B
	33AA6FB3	B19B03E0	6BD8AFDF	D0452AA2	93A77FEE
	18A6A41A	ED3ADAEC	C8D8F4F2	5C2DD7B1	3FCDF8B6
	C0B4C926	73BF924B	50D498DA	2A16E373	DD405AD1
	10BEA013	7317B308			
Y_0	E532CF6E	17E192B6	94E1313A	3CE04353	4CD24E26

Parameter	Hexadecimal Value				
	4E773F7B	DE8D9A19	0EF1A5AB	841C4D39	32A85B48
	D428875D	D7101764	89E6973D	CBB9DC37	9432C26B
	33AA6FB3	B19B03E0	6BD8AFDF	D0452AA2	93A77FEE
	18A6A41A	ED3ADAEC	C8D8F4F2	5C2DD7B1	3FCDF8B6
	C0B4CAD1	DD7272F5	264C5D86	5ED8F6D2	D2796221
	88BF4813	601FC040			
p	E532CF6E	17E192B6	94E1313A	3CE04353	4CD24E26
	4E773F7B	DE8D9A19	0EF1A5AB	841C4D39	32A85B48
	D428875D	D7101764	89E6973D	CBB9DC37	9432C26B
	33AA6FB3	B19B03E0	6BD8AFDF	D0452AA2	93A77FEE
	18A6A41A	ED3ADAEC	C8D8F4F2	5C2DD7B1	3FCDF8B6
	C0C8A33E	72998487	DE550E0D	7188AA87	B2CF20D5
	C7F83EB0	C95ED0C1			

Generation of *q*

Parameter	Hexadecimal Value				
X_{q1}	1F2654CE	23E5F777	0F872867	0C	
X_{q2}	1D867E64	065DEE3E	7955F2EB	C7	
q_1	1F2654CE	23E5F777	0F872867	25	
q_2	1D867E64	065DEE3E	7955F2EB	DF	
R_q	B9B12BAA	FCDC8B84	BD9F7A1A	D95841D0	FFD0EFE8
	774C9E96	4F			
X_q	D2DDB927	5760C8AA	CFC46AD6	C3D78F07	88D1873
	3D6B0240	43E04AE7	0E71794C	0231ACBA	D28FC0F
	C137D8CF	A8487475	94F6EE8D	5D318C69	62A70AB4
	71BA439B	1227EEEA	FDEAB75A	C119D721	78BAF468
	5B931DF1	C87E230F	33EB120C	DFE7A618	A48C63B2
	069E6261	0152752C	113990A2	1A551FE1	7F6F711C
	E812197E	1C9CE5DE	A6		
Y_0	D2DDB927	5760C8AA	CFC46AD6	C3D78F07	88D18733
	D6B02404	3E04AE70	E71794C0	231ACBAD	28FC0FC1
	37D8CFA8	48747594	F6EE8D5D	318C6962	A70AB471
	BA439B12	27EEEAFD	EAB75AC1	19D72178	BAF4685B
	931DF1C8	7E230F33	EB120CDF	E7A618A4	8C63B206
	9E626348	83495D14	2AD86A3C	1B342BE9	84635E1F
	11B1A62E	764F0CF1			
q	D2DDB927	5760C8AA	CFC46AD6	C3D78F07	88D18733
	D6B02404	3E04AE70	E71794C0	231ACBAD	28FC0FC1
	37D8CFA8	48747594	F6EE8D5D	318C6962	A70AB471
	BA439B12	27EEEAFD	EAB75AC1	19D72178	BAF4685B
	931DF1C8	7E230F33	EB120CDF	E7A618A4	8C63B206
	9E6CCCFF	4653797D	04279F67	40FC3A73	50A196A9
	AC22D459	188697F3			

Generation of *d* and *n*

Parameter	Hexadecimal Value				
d	7DDC2086	E1AB2836	9A3B3D1F	FF3FED9C	3E1C5090
	26469638	500A676F	C8DEF9EE	D45FC640	1BD58942
	A1D8756A	9DD47CD8	A81F9507	2F128B42	1A5FF599
	06F511D6	D28A7731	1C74BC83	38E4BC39	37CBF2CE
	35DF7890	A5061793	E2792DEC	A294BA40	BBE610A1
	E23F003C	ECA823FF	43FE0503	856DD102	F3D6443E
	CE6CBAF7	D4F16DFD	BAF7C2F8	CB4955E2	F2DF29FD
	A7A4DD7B	93785252	E2CA4AAD	14E8B1A1	7881D381
	DAD9061E	C8AB50DD	CBF8B56A	FA464F01	D28DFDFF
	0CAFF6B5	588E8F77	6FA128DE	7C102494	7D5D8067
	3F545A5C	73FBC16B	609A33A1	1021DD8F	37A4E7C4
	78AA19E3	BCB1BBD8	F4AA9232	65F1F1D5	5236753A
	C8C10D60	F0FAB073	66439A90	5E2C63AB	
n	BCCA30CA	5280BC51	E758DBAF	FEDFE46A	5D2A78D8
	3969E154	780F9B27	AD4E76E6	3E8FA960	29C04DE3
	F2C4B01F	ECBEBB44	FC2F5F8A	C69BD0E3	278FF065
	8A6F9AC2	3BCFB2C9	AAAF1AC4	D5571A55	D3B1EC35
	50CF34D8	F789235D	D3B5C4E2	F3DF1761	19D918F2
	D35E805B	62FC35FE	E5FD0785	4824B984	6DC1665E
	35A31873	BF6A24FE	50842D0A	A0305C35	D0F45B0D
	7C2F1E94	32D850D6	7956D383	BBEF52FC	2ACBF7AE
	6F7CA214	88A56456	BDF66726	96EE037C	3CAA2199
	904E37AA	40134E10	155FC813	93A225BD	129C4B3B
	C91AD3A5	FC958A6A	BCABE355	0390B677	87625D78
	F8D3172B	673C4482	CE354B89	51D7E8C4	DDCE5D4C
	DFA6790C	6CE8C02C	8D807AE2	6F27FE33	

D.3.2 Signature Generation

Parameter	Hexadecimal Value				
M=abc	616263				
H(M)	A9993E36	4706816A	BA3E2571	7850C26C	9CD0D89D

The string x'33CC' is postfixed to the hash, indicating that the hash function is SHA-1. Padding consisting of repetitions of the nibble x'B' is prefixed to the hash. This is preceded by the Header hex value x'6'.

Parameter	Hexadecimal Value				
IR	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBAA999	3E364706
	816ABA3E	25717850	C26C9CD0	D89D33CC	

Parameter Hexadecimal Value

For odd e, ***IR*** = ***RR***, thus the value $\boldsymbol{RR}^d$ mod n is computed, and since this value is less than $n/2$, the signature $\Sigma = \boldsymbol{RR}^d$ mod n.

Parameter					
Σ	2B655743	4AF5230B	2693648C	2C797A47	E197DA08
	D4AEF6F3	D49EDFA2	2E7332A2	27366F2B	C597E0D5
	4863C84E	9139AEAD	C9D17460	4119FAD9	59B2BF8E
	DDFB15CA	6B06C3EF	D7F6A740	719E541D	28983384
	5F5AE388	8003711E	CB1027CF	8600C20A	37199035
	92C63533	AB7258EB	A709F5F4	5B8E9DF3	3532CC70
	D9165BE2	AA7BCE29	9C951D58	5F74C3FC	25685788
	ABC33257	14FDF593	3CA6AC52	8EBBEAD9	9E02DC79
	C1151156	BC59DBB8	F32BDFAB	92F7B28F	B35C646A
	281CDD75	460F0743	6A140BCB	47A971CF	157D0773
	888B9C07	723E7749	22FCDFD2	04CD4BC3	17B4D5F6
	EF6F4CB4	E6840BB5	6E2DDEF8	8C349E8F	D9CBA049
	D1DF2329	848A7F42	27E44021	813B7F8D	

D.3.3 Signature Verification

Parameter Hexadecimal Value

The value $(\Sigma')^e$ mod n is computed to yield ***RR′***.

Parameter					
RR′	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBAA999	3E364706
	816ABA3E	25717850	C26C9CD0	D89D33CC	

Since ***RR′*** is congruent to 12 mod 16, the recovered hash ***IR′*** is set equal to ***RR′***. Since ***IR′*** is identical to the computed hash ***IR***, and the signature verification is successful.

D.4 Odd *e* = 3 with 4096-bit *n*

D.4.1 Key Generation

Generation of *p*

Parameter	Hexadecimal Value			
X_{p1}	1A1916DD	B29B4EB7	EB6732E1	28

Parameter	Hexadecimal Value				
X_{p2}	192E8AAC	41C576C8	22D93EA4	33	
p_1	1A1916DD	B29B4EB7	EB6732E1	5B	
p_2	192E8AAC	41C576C8	22D93EA4	49	
R_p	232C6DE7	2FCAF2EA	01B5E88F	6BCB1F7E	6BDD6618
	47A83879	02D			
X_p	CA81AA2B	3D94CD01	37D841AE	F64F6C57	1C92C06C
	ADFACE7B	E2D6B173	5BDB57A6	48302E58	F11A0BD0
	04E5CC30	D594FBBD	1C060B07	F9F93323	1434BEDE
	B8951A2E	BB9B7E79	1AAD732E	A111F776	73B1D187
	3DAEE1C9	BB671030	644E9E3B	5B9B05AB	A3E61A05
	9C9A5D57	A51CDA94	8DC49300	1BE4C8E5	0A53A00A
	BAF59618	2AEEDDB4	24CD56AC	4B4BCAA8	FF6E713B
	A0B1893D	FF17CB13	5B769F60	044750BF	A23FC73D
	D7B8F51E	EAC09C74	8CEEAAFE	1B884589	F2722166
	57E6EB5B	D9A0AFE6	F2D4E477	B54CE17F	85032D4B
	4DD3DC34	E315C217	5261EF6F	9B3E3124	618CFD39
	79DD1D02	B21BBD18	004E811B	C49D2EAC	DCB1EBE3
	F8374DFB	A2D55364	54F33F52	2D894581	
Y_0	CA81AA2B	3D94CD01	37D841AE	F64F6C57	1C92C06C
	ADFACE7B	E2D6B173	5BDB57A6	48302E58	F11A0BD0
	04E5CC30	D594FBBD	1C060B07	F9F93323	1434BEDE
	B8951A2E	BB9B7E79	1AAD732E	A111F776	73B1D187
	3DAEE1C9	BB671030	644E9E3B	5B9B05AB	A3E61A05
	9C9A5D57	A51CDA94	8DC49300	1BE4C8E5	0A53A00A
	BAF59618	2AEEDDB4	24CD56AC	4B4BCAA8	FF6E713B
	A0B1893D	FF17CB13	5B769F60	044750BF	A23FC73D
	D7B8F51E	EAC09C74	8CEEAAFE	1B884589	F2722166
	57E6EB5B	D9A0AFE6	F2D4E477	B54CE17F	85032D4B
	4DD3DC34	E315C217	5261EF6F	9B3E3124	618CFD39
	79DD1D02	B21BBD18	004E836C	393D5C08	0209A59B
	F0BA2A45	372605F0	827BD16E	3A3DE4E3	
p	CA81AA2B	3D94CD01	37D841AE	F64F6C57	1C92C06C
	ADFACE7B	E2D6B173	5BDB57A6	48302E58	F11A0BD0
	04E5CC30	D594FBBD	1C060B07	F9F93323	1434BEDE
	B8951A2E	BB9B7E79	1AAD732E	A111F776	73B1D187
	3DAEE1C9	BB671030	644E9E3B	5B9B05AB	A3E61A05
	9C9A5D57	A51CDA94	8DC49300	1BE4C8E5	0A53A00A
	BAF59618	2AEEDDB4	24CD56AC	4B4BCAA8	FF6E713B
	A0B1893D	FF17CB13	5B769F60	044750BF	A23FC73D
	D7B8F51E	EAC09C74	8CEEAAFE	1B884589	F2722166
	57E6EB5B	D9A0AFE6	F2D4E477	B54CE17F	85032D4B
	4DD3DC34	E315C217	5261EF6F	9B3E3124	618CFD39
	79DD1D02	B21BBD18	0057B815	158484E9	1B8343AF
	E894241D	73BF1D97	D0D66745	AB5A8045	

Generation of *q*

Parameter	Hexadecimal Value				
$\mathbf{X}_{q1}$	1FA10E5B	D0CE77C7	69161F48	51	
$\mathbf{X}_{q1}$	FD0872DE	8673BE3B	48B0FA42	8	
$\mathbf{X}_{q2}$	18A572DC	72EE1222	919F1632	68	
q_1	1FA10E5B	D0CE77C7	69161F48	59	
q_1	FD0872DE	8673BE3B	48B0FAC2	9	
q_2	18A572DC	72EE1222	919F1632	85	
$\mathbf{R}_q$	257FCFDA	63F09655	8AD4F66D	55627777	684433B3
	DC017781	25A			
$\mathbf{R}_q$	2035085B	AD04DFA2	D88AFDBF	5CEACD9D	3F27739C
$\mathbf{X}_q$	B6C6E702	AC20A71D	A8787858	8473FF71	7674ED24
	B7592ED0	70DA938A	B752B490	CE4E73A0	53D90CB9
	1FE75727	522C2B22	C66CA591	C4D1AF61	C36582B4
	B4E41834	7305F685	6BE5E65A	6A6FCE86	5163E57B
	DD38F3BB	6E5C1D03	5DD15056	954E5B21	9EA64C82
	D5056420	73DC7A7C	A8D49569	8C9EFF58	FBEA4E97
	96539B07	1C8CE95E	01C6F92A	C82CEE15	427FA42F
	51EE3A09	F9986567	B6386395	E56FC5EE	5E77D8F3
	9C6BEFF3	490D7C3A	CBED1584	B8D7C915	9F45EB43
	7ED48F3E	F2481B56	C4756001	1412D6E1	3E3E7D00
	7B5C05DB	F5FF0D0F	CFF1FA20	70D16C7A	BA93EDFB
	35D87005	67E5913D	B734E3FB	D15862EB	C59FA042
	5DFA131E	5491379E	876B7778	21BFDD48	
$\mathbf{Y}_0$	B6C6E702	AC20A71D	A8787858	8473FF71	7674ED24
	B7592ED0	70DA938A	B752B490	CE4E73A0	53D90CB9
	1FE75727	522C2B22	C66CA591	C4D1AF61	C36582B4
	B4E41834	7305F685	6BE5E65A	6A6FCE86	5163E57B
	DD38F3BB	6E5C1D03	5DD15056	954E5B21	9EA64C82
	D5056420	73DC7A7C	A8D49569	8C9EFF58	FBEA4E97
	96539B07	1C8CE95E	01C6F92A	C82CEE15	427FA42F
	51EE3A09	F9986567	B6386395	E56FC5EE	5E77D8F3
	9C6BEFF3	490D7C3A	CBED1584	B8D7C915	9F45EB43
	7ED48F3E	F2481B56	C4756001	1412D6E1	3E3E7D00
	7B5C05DB	F5FF0D0F	CFF1FA20	70D16C7A	BA93EDFB
	35D87005	67E5913D	B734E930	58AFE3DB	D44DE427
	50646158	A8651ED4	810DC990	ED8947F5	
	35D87005	67E5913D	B734E54A	E0ACBF5C	3AF47C65
q	B6C6E702	AC20A71D	A8787858	8473FF71	7674ED24
	B7592ED0	70DA938A	B752B490	CE4E73A0	53D90CB9
	1FE75727	522C2B22	C66CA591	C4D1AF61	C36582B4
	B4E41834	7305F685	6BE5E65A	6A6FCE86	5163E57B
	DD38F3BB	6E5C1D03	5DD15056	954E5B21	9EA64C82
	D5056420	73DC7A7C	A8D49569	8C9EFF58	FBEA4E97
	96539B07	1C8CE95E	01C6F92A	C82CEE15	427FA42F
	51EE3A09	F9986567	B6386395	E56FC5EE	5E77D8F3
	9C6BEFF3	490D7C3A	CBED1584	B8D7C915	9F45EB43
	7ED48F3E	F2481B56	C4756001	1412D6E1	3E3E7D00

Parameter	Hexadecimal Value				
	7B5C05DB	F5FF0D0F	CFF1FA20	70D16C7A	BA93EDFB
	35D87005	67E5913D	B7513E91	95D19872	2FECAAEA
	6969FA28	9B742A88	8CDA1F83	D5B10F8B	
	35D87005	67E5913D	B75E0A39	2143E629	ACA7C985

<u>Generation of *d* and *n*</u>

Parameter	Hexadecimal Value				
d	1818EBAA	E97DC914	41EC3291	DDBDECFA	D2FA18CD
	7D837A70	3566E914	033D571C	5FE8462E	4DC90CB6
	1F5EC2E5	DB008810	EFD4E0B4	59D0C997	68D5DA7A
	D56F1DA5	10E23E1D	04CBB7DD	4756C3C1	34484FFA
	D30194F5	D323D036	14816330	4B1B636B	DEE0CCE0
	CA06692F	7015113A	BA03242A	A77CE321	DC9BE825
	849C81D3	333EBC96	E123D70E	2440F921	BA9F588D
	8B67B05B	056B09C6	252624C7	E89E5755	6349CF32
	7BC42CC6	4724514A	227B181F	1AEE21E9	EEF500A6
	6EF92065	E29E3ECD	8AA52A8A	8B514859	E7BC2EC8
	DFCD32A1	ACFB2846	3FDBF70D	44831BDE	D62D2AFD
	ABE8B9A7	45A94688	39E68C23	1B21D1DE	2C888717
	A7A398E3	2C3E0299	66C6DDE3	446B1AC2	5ACC1129
	A42123A1	67D89076	D334D179	B64FC52B	735E6C50
	19DB868A	2883E7B4	66287D59	DC426EAD	974067A9
	D1E3845B	4B459370	8FB8BE88	C4160DD3	CF2563F6
	6E349483	0F89CD16	082885DF	CD8236DB	E3EAA0BE
	F086D3FE	429CA248	0B2A2BD9	73A030E9	D8B4D3AE
	344E7D75	0BD39D95	FBC172B5	F5256E97	BD8C0753
	7308A3F8	D3F874EF	2566F5FB	3207512A	5EC8A920
	CB0A4B56	994E1A57	DEC4BCA7	3E5A0BE2	AF8D1373
	87F6FA4A	B4146B8D	26B6C5E0	1EDCEFF5	65ADFFE9
	B8A41730	D0852AD0	A384690A	FCDCDCFD	CAF6D7A5
	BF4B070C	9ABACB3E	D9FABC5B	FC2E33EC	2AFC5E41
	BE134A63	E1B4AB7B	A2997DB6	584E8170	FAD3D6CF
	BC476FD4	6D8510F2	A7E2DAC7		
	ABE8B9A7	45A94688	39E83BFF	FE9BD223	B42FDCDA
n	90958601	78F2B679	8B892F6B	32738DE0	F1DC94D0
	F114DEA1	40697678	13700AAA	3F71A515	D2B64C44
	BC389163	22033065	9EFD443A	1AE4B98C	75031EE1
	009AB1DE	654D74AE	1CC64F2F	AC089687	39B1DFE0
	F2097DC2	F2D6E144	7B085321	C2A45487	3944CD44
	BC26771C	A07E6760	5C12D8FF	ECED52CB	2BA770E1
	1BAB0AF3	33786B89	46D70A54	D985D6CA	5FBC1351
	446E2222	20823AA4	DEE4DCAF	73B60C00	53BADB2E
	E6990CA5	AAD9E7BC	CEE290BA	A194CB7B	99BE03E6
	99D6C263	4FB578D1	3FDEFF3F	43E7B21B	6E6918B5
	3ECF2FCA	0DE2F1A5	7F27CA4F	9B12A739	050F01F2
	077459EB	A1F7A731	5B6748D2	A2CAEB35	0B332A8D
	EDD59553	09740F98	68A93353	9A82A08F	A210F827
	C27C49E7	4F641CD0	6E0054A2	D8E64C96	198A872C
	EED66C3B	06457A71	7B719214	6E81B09A	B04F9153
	13164103	A614253D	1D1F59B9	701E948A	48598A29
	C3DCF010	E3CE280D	3C74E93B	9623002A	726799FE
	CCEC2529	51CBBC42	33E667E5	F84D8C03	85DCB78D

Parameter	Hexadecimal Value				
	52D045CF	7D8ED9ED	8F0C7881	C51E8630	C2915D13
	F9AF9EE7	1E670D71	F3E27CA1	6E19FC69	2B53BA0C
	BAEDF482	A983A105	22538299	76D3E781	917359C7
	6397F66F	915645D1	BCA8B1E0	4AE5AC6A	38CF7A15
	1FC15662	9A69455C	9E7A2EA2	B06ED83E	8AF8EFF3
	54D6F972	C2B4AD09	27F007C7	053622BD	B19FC292
	8E750CAD	01E4FB8C	7AEF0FA1	5D46F740	32F52724
	78DFE71A	EECEEC79	705CB077		
	077459EB	A1F7A731	5B7167FF	F7A6ECD6	391F2D20

D.4.2 Signature Generation

Parameter	Hexadecimal Value				
M=abc	616263				
H(M)	A9993E36	4706816A	BA3E2571	7850C26C	9CD0D89D

The string x'33CC' is postfixed to the hash, indicating that the hash function is SHA-1. Padding consisting of repetitions of the nibble x'B' is prefixed to the hash. This is preceded by the Header hex value x'6'.

IR	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBAA999	3E364706	816ABA3E
	25717850	C26C9CD0	D89D33CC		

For odd e, ***RR*** = ***IR***. The value ***RR***^{d} mod n is computed as:

For odd e, ***RR*** = ***IR***. The value ***RR***^{d} mod n is now computed, and since this value is greater than $n/2$, the signature $\Sigma = n - (\boldsymbol{RR}^{d} \bmod n)$.

Parameter	Hexadecimal Value				
	7DBAD001	FABDF0EB	5558B5FA	874D5239	27EBA36F
	BBAF6EBF	57EB5BA2	7756B664	0885A988	F4DD66E3
	8BD59E7B	474224E8	25FEB74E	C682C8E4	E7A1B468
	61880F30	DC10D0DA	D87C7831	97EC80D5	74DFDA80
	99BCFA66	B13377EA	817B821F	A739A991	1B0586A2
	FA4E135E	C489097A	6C7DF125	748FF80B	0F67B240
	B61089CE	9B90D761	9D9700F4	437BFD5B	9687BEBF
	8FBC9925	C1991E4B	F4571A15	F042E90C	AC5119DA
	593F90C5	D3EDAB2D	BD9B4DA0	E87A8324	DD8F6B10
	9178C199	B7C8CC52	2B9FAA5F	65606F68	C11F80CE
	949A6B86	2F3B20CB	FB644101	63F48F6F	9A5904B5
	86DF32FC	17E98B84	B60652B1	2C173571	EE4451FD
	9C781451	01123515	5165FF0A	3494E6C3	C3BEF977
	28B0CAB8	AC616320	0D13E0D3	FA9C0A8D	622D0F4E
	76C8B3B5	35759557	C65DCE75	E35E9E3C	91C512B1
	4AB2C1A2	31EA17C0	98CEC30F	1332A2DF	74204AE1
	5396A89A	2DA4AD47	CA69E6C4	373D7266	60DEA1DF
	F3B34848	80294648	F89549DB	81AFFCDA	43F676EA
	2A7F115D	B283C2A6	7F9B3E68	E4497C49	CACFBEBB
	340B0AAD	6B4E56E0	5AC0B2A3	09F5060F	E9061E1C
	47B86A25	DDD533D2	DB25C85A	F6B333DD	7E53F2F8
	F5588727	7AED499F	4EF14DF7	F7C8FC55	4982DAC0
	515C9659	BD30854B	F5B84AF9	DC85787F	8E7DF23A
	CE7A3652	D1C1D496	6E5F0E9B	62342CEF	05558D51
	480ADED0	923BF63F	0C772C62	B7615263	4AB2E14D
	45DED221	AE7104C2	913EFF5E		

now computed, and since this value is greater than $n/2$, the signature $\Sigma = n - (\boldsymbol{RR}^d \bmod n)$.

Parameter	Hexadecimal Value				
Σ	12DAB5FF	7E34C58E	36307970	AB263BA7	C9F0F161
	35656FE1	E87E1AD5	9C195446	36EBFB8C	DDD8E561
	3062F2E7	DAC10B7D	78FE8CEB	5461F0A7	8D616A78
	9F12A2AD	893CA3D3	4449D6FE	141C15B1	C4D20560
	584C835C	41A36959	F98CD102	1B6AAAF6	1E3F46A1
	C1D863BD	DBF55DE5	EF94E7DA	785D5AC0	1C3FBEA0
	659A8124	97E79427	A9400960	9609D96E	C9345491
	B4B188FC	5EE91C58	EA8DC299	837322F3	A769C154
	8D597BDF	D6EC3C8F	11474319	B91A4856	BC2E98D6
	085E00C9	97ECAC7F	143F54DF	DE8742B2	AD4997E6
	AA34C443	DEA7D0D9	83C3894E	371E17C9	6AB5FD3C
	809526EF	8A0E1BAC	A560F621	76B3B5C3	1CEED890
	515D8102	0861DA83	17433449	65EDB9CB	DE51FEB0
	99CB7F2E	A302B9B0	60EC73CE	DE4A4208	B75D77DE
	780DB885	D0CFE519	B513C39E	8B23125E	1E8A7EA1
	C8637F61	742A0D7C	845096AA	5CEBF1AA	D4393F48
	70464776	B6297AC5	720B0277	5EE58DC4	1188F81E
	D938DCE0	D1A275F9	3B511E0A	769D8F29	41E640A3
	28513471	CB0B1747	0F713A18	E0D509E6	F7C19E58
	C5A49439	B318B691	9921C9FE	6424F659	424D9BF0
	73358A5C	CBAE6D32	472DBA3E	8020B3A4	131F66CE
	6E3F6F48	1668FC32	6DB763E8	531CB014	EF4C9F54

Parameter	Hexadecimal Value				
	CE64C008	DD38C010	A8C1E3A8	D3E95FBE	FC7AFDB8
	865CC31F	F0F2D872	B990F92B	A301F5CE	AC4A3541
	466A2DDC	6FA9054D	6E77E33E	A5E5A4DC	E84245D7
	330114F9	405DE7B6	DF1DB119		

D.4.3 Signature Verification

Parameter Hexadecimal Value

The value $(\Sigma')^e$ mod n is computed to yield ***RR′***.

Parameter	Hexadecimal Value				
RR′	7DBAD001	FABDF0EB	5558B5FA	874D5239	27EBA36F
	BBAF6EBF	57EB5BA2	7756B664	0885A988	F4DD66E3
	8BD59E7B	474224E8	25FEB74E	C682C8E4	E7A1B468
	61880F30	DC10D0DA	D87C7831	97EC80D5	74DFDA80
	99BCFA66	B13377EA	817B821F	A739A991	1B0586A2
	FA4E135E	C489097A	6C7DF125	748FF80B	0F67B240
	B61089CE	9B90D761	9D9700F4	437BFD5B	9687BEBF
	8FBC9925	C1991E4B	F4571A15	F042E90C	AC5119DA
	593F90C5	D3EDAB2D	BD9B4DA0	E87A8324	DD8F6B10
	9178C199	B7C8CC52	2B9FAA5F	65606F68	C11F80CE
	949A6B86	2F3B20CB	FB644101	63F48F6F	9A5904B5
	86DF32FC	17E98B84	B60652B1	2C173571	EE4451FD
	9C781451	01123515	5165FF0A	3494E6C3	C3BEF977
	28B0CAB8	AC616320	0D13E0D3	FA9C0A8D	622D0F4E
	76C8B3B5	35759557	C65DCE75	E35E9E3C	91C512B1
	4AB2C1A2	31EA17C0	98CEC30F	1332A2DF	74204AE1
	5396A89A	2DA4AD47	CA69E6C4	373D7266	60DEA1DF
	F3B34848	80294648	F89549DB	81AFFCDA	43F676EA
	2A7F115D	B283C2A6	7F9B3E68	E4497C49	CACFBEBB
	340B0AAD	6B4E56E0	5AC0B2A3	09F5060F	E9061E1C
	47B86A25	DDD533D2	DB25C85A	F6B333DD	7E53F2F8
	F5588727	7AED499F	4EF14DF7	F7C8FC55	4982DAC0
	515C9659	BD30854B	F5B84AF9	DC85787F	8E7DF23A
	CE7A3652	D1C1D496	6E5F0E9B	62342CEF	05558D51
	480ADED0	923BF63F	0C772C62	B7615263	4AB2E14D
	45DED221	AE7104C2	913EFF5E		

Since ***RR′*** is congruent to 6 mod 8, the recovered hash ***IR′*** is set equal to *n*-***RR′***.

Parameter	Hexadecimal Value				
IR′	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
RR′	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB

Parameter	Hexadecimal Value				
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBAA999	3E364706	816ABA3E
	25717850	C26C9CD0	D89D33CC		

Since ***IR′*** is identical to the computed hash ***IR***, the signature verification is successful.

D.5 Even *e* =2 with 1024-bit *n*

D.5.1 Key Generation

<u>Generation of *p*</u>

Parameter	Hexadecimal Value				
X_{p1}	1857FA8D	9D0B0E4E	033B68CE	40	
X_{p2}	1A416A5C	0DAB5E4C	EB97EB9E	EC	
p_1	1857FA8D	9D0B0E4E	033B68CE	4D	
p_2	1A416A5C	0DAB5E4C	EB97EB9F	45	
R_p	99ED610E	496D79B3	885ADE40	A1DD6BF9	887D0406
	C84E64E0	B6			
X_p	DBB3CB4C	375C0ECD	2FD300DB	4F085472	93CA004C
	EDD2019C	E79CA08A	15EEFB25	DD3BAF98	1823961F
	4148FF03	C6E96930	B78C7A75	8D69BC50	70137900
	403807A4				
Y_0	DBB3CB4C	375C0ECD	2FD300DB	4F085472	93CA004C
	EDD2019C	E79CA08A	15EEFB25	DD3BAF98	1823979D
	2C647C5D	6A343A75	B0254CFA	7ABE79C1	5CA3878A
	1AACA80E				
p	DBB3CB4C	375C0ECD	2FD300DB	4F085472	93CA004C
	EDD2019C	E79CA08A	15EEFB25	DD3BAF98	183B0C2F
	01D7F8B4	931856F6	DD3EBA17	7D763C03	E1DCEABC
	D803BE33				

<u>Generation of *q*</u>

Parameter	Hexadecimal Value			
X_{q1}	198C4AD4	DBA81A56	0183DD19	BA

Parameter	Hexadecimal Value				
X_{q2}	1A5A797E	B98BEFF4	312D0C9E	F7	
q_1	1A5A797E	B98BEFF4	312D0C9E	F7	
q_2	1A5A797E	B98BEFF4	312D0C9F	5B	
R_q	8B40EE98	A4319A3A	6426B5DB	25915F92	14CCE454
	595E015F	17			
X_q	EEAA4A53	47999FE7	6FB78760	64BBEC66	CB409A77
	39EF5A59	06613DC3	7225D41D	2BEB1F9F	5EBF4CC9
	5B6BF916	A47C5EF2	146BE00E	CD4A1C5D	88B3E85A
	9569BE97				
Y_0	EEAA4A53	47999FE7	6FB78760	64BBEC66	CB409A77
	39EF5A59	06613DC3	7225D41D	2BEB1F9F	5EBF4D0C
	4F838414	B1ACDC53	E029BDFF	9E808D35	454801F5
	78A5466B				
q	EEAA4A53	47999FE7	6FB78760	64BBEC66	CB409A77
	39EF5A59	06613DC3	7225D41D	2BEB1F9F	5EC77A85
	38767A87	BB7015D6	07FF26DE	61282753	9306BA1C
	FFF093A7				

Generation of *d* and *n*

Parameter	Hexadecimal Value				
d	199A6985	E9B2BFF5	A2841CCC	D80FC73A	28A14266
	0987EB12	3DBCAEB2	B8EE546D	2356A3A5	7D9C28ED
	71E455C4	466CBE30	7787DC5A	9959B747	5A189A8F
	038A4741	E4B10153	BE08C26E	4401F7AB	6E7E9609
	2CAF07C0	870B13B6	4F669667	3029EC2C	77AABC39
	7FA528A2	45D7073C	E69CC9BD	CD7BEF91	599DCA48
	4000C0BD	8AB0814E			
n	CCD34C2F	4D95FFAD	1420E666	C07E39D1	450A1330
	4C3F5891	EDE57595	C772A369	1AB51D2B	ECE1476B
	8F22AE22	3365F183	BC3EE2D4	CACDBA3A	D0C4D478
	1C523A10	EFE6203D	6F3BC226	BF9A4597	27B8F122
	C482D8C8	6019F9A8	69329187	096430A6	C67CB103
	742BCBC6	6906AD23	836EBABB	511D5D80	AB8CB599
	74E9AAC6	2D785C45			

D.5.2 Signature Generation

Parameter	Hexadecimal Value				
M=abc	616263				
H(M)	A9993E36	4706816A	BA3E2571	7850C26C	9CD0D89D

The string x'33CC' is postfixed to the hash, indicating that the hash function is SHA-1. Padding consisting of repetitions of the nibble x'B' is prefixed to the hash. This is preceded by the Header hex value x'6'.

IR	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBAA999	3E364706	816ABA3E	25717850
	C26C9CD0	D89D33CC			

Since the Jacobi symbol $\left(\frac{\boldsymbol{IR}}{n}\right)$ of the encapsulated hash with respect to n is -1, the encapsulated hash ***IR*** is divided by 2 before signing in order to force the Jacobi symbol to +1.

RR	35DDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD
	DDDDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD
	DDDDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD
	DDDDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD
	DDDDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD	DDDDDDDD
	DDDDDDDD	DDDD54CC	9F1B2383	40B55D1F	12B8BC28
	61364E68	6C4E99E6			

The value $\boldsymbol{RR}^{d}$ mod n is computed. This is less than $n/2$, thus the signature $\Sigma = \boldsymbol{RR}^{d}$ mod n.

Parameter	Hexadecimal Value				
Σ	232F0E08	EB9A2395	7646697F	C7884796	D39A04FD
	0EFF5B72	B60813D4	E6919178	91C96603	876D0879
	3AAD86DA	F2E6187F	F62C226E	81BD6B99	3B27091E
	0864895A	F10F222A	EB022961	B444D312	EA3DB789
	1D4550B2	80CF2469	3D4465B9	57E53CBD	B0F8C29D
	2B5EE154	5D6C91A4	5EAAACEC	0096D8A5	E4CFE06A
	2CD320BD	F853D817			

D.5.3 Signature Verification

Parameter Hexadecimal Value

The value $(\Sigma')^e$ mod n is computed to yield ***RR′***.

Parameter	Hexadecimal Value				
RR′	96F56E51	6FB821CF	36430888	E2A05BF3	672C3552
	6E617AB4	100797B7	E994C58B	3CD73F4E	0F03698D
	B144D044	558813A5	DE6104F6	ECEFDC5C	F2E6F69A
	3E745C33	1208425F	915DE448	E1BC67B9	49DB1344
	E6A4FAEA	823C1BCA	8B54B3A9	2B8652C8	E89ED325
	964DEDE8	8B295856	E4539738	10680061	98D3F971
	13B35C5D	C129C25F			

Since ***RR′*** is congruent to 7 mod 8, the recovered encapsulated hash ***IR’*** is computed as 2(n - ***RR′***).

Parameter	Hexadecimal Value				
IR′	6BBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB	BBBBBBBB
	BBBBBBBB	BBBAA999	3E364706	816ABA3E	25717850
	C26C9CD0	D89D33CC			

Since the recovered hash ***IR’*** is identical to the computed hash ***IR***, the signature verification is successful.

Appendix E: Implementation Considerations

(Informative)

E.1 Fast Signature Algorithm

This Section presents an efficient algorithm for computing the signature, Σ, using the Chinese Remainder Theorem (CRT). It makes use of the additional quantities p, q, d mod (p-1), d mod (q-1), and q^{-1} mod p (where p is the larger of the two primes). The last three quantities may be pre-computed during the key generation process. These additional quantities are subject to the same protection (confidentiality and integrity) requirements as the private key d (see Section 4.1.4).

The quantities p, q, n, e, and d are as defined in Section 0, *Key Generation*. The signature Σ is computed from the integer signature block x as follows, where p is the larger of the two primes, and $u = q^{-1} \bmod p$. p_2 and q_2 are temporary variables.

1. Set $p_2 = ((x \bmod p)^{d \bmod (p-1)}) \bmod p$.
2. Set $q_2 = ((x \bmod q)^{d \bmod (q-1)}) \bmod q$.
3. If $p_2 = q_2$, then set $\Sigma = p_2$; stop.
4. Set $p_2 = p_2 - q_2 \bmod p$.
5. If ($p_2 < 0$) then set $p_2 = p_2 + p$.
6. Set $\Sigma = q_2 + (q((p_2 u) \bmod p))$.

Unlike the signature computed in Section 0, *Signature Generation*, (recall that the signature is defined as $\Sigma = \min \{ RR^d \bmod n, n-(RR^d \bmod n) \}$), the signature calculated here using the CRT is always Σ, and never its complement, n-Σ

NOTE
RSA implementations using the Chinese Remainder Theorem are susceptible to fault analysis. Refer to Appendix C.5 *Cryptographic Calculation Errors* for further information.

E.2 Multiplicative Inverse

To compute e^{-1}, the multiplicative inverse of e mod n, with $0 < e < n$, and where e and n are relatively prime, that is, GCD(e, n) = 1, perform the following steps:

Step 1: Set $i = n$, $h = e$, $v = 0$ and $d = 1$.

Step 2: While $h > 0$, repeat steps 3 through 5.

Step 3: $t = i$ DIV h where DIV is defined as integer division.

Step 4: Set $x = h$, $h = i - tx$ and $i = x$.

Step 5: Set $x = d$, $d = v - tx$ and $v = x$.

Step 6: $e^{-1} = v \bmod n$.

For examples of Greatest Common Divisor (GCD) algorithms, see [6].

E.3 Sieving

A *sieve procedure* is described as follows: Given a sequence of integers Y_0, Y_0+1, …, Y_0+J, a sieve will identify which integers in the sequence are divisible by small primes.

Start by selecting a *factor base* of all the primes p_j, from 2 up to some selected limit L. Compute $S_j = Y_0 \bmod p_j$ for all p_j in the factor base. Initialize an array of length J+1 to zero. Then starting at $Y_0 - S_j + p_j$, let every p_j^{th} element of the array be set to 1. Do this for the entire length of the array and for every j.

When finished, every location in the array which has the value 1, is divisible by some small prime and is hence composite.

The array can be either a bit array for compactness when memory is small, or a byte array for speed when memory is readily available. There is no need to sieve the entire sieve interval at once. The array can be partitioned into suitably small pieces, sieve each piece, then go on to the next piece. When finished, every location with the value 0 is a candidate for prime testing.

The amount of work for this procedure is approximately $M \log \log L$, where M is the length of the sieve interval; this is a very efficient procedure for removing composite candidates for primality testing.

NOTE

The prime '2' takes the longest amount of time ($M/2$) to sieve since it touches the most locations in the sieve array. An easy optimization is to combine the initialization of the sieve array with the sieving of the prime '2'. With a little clever coding, the sieving of the prime '3' can be included during initialization. This can save 1/3 of the total sieve time.

E.4 Fast Prime Generation

Two strong primes, each of size 512+128*s* bits, are randomly generated in such a way that their product is 1024+256*s* bits. The procedure for prime generation that is outlined below is therefore executed twice; once for p and once for q. The procedure refers to p only.

Summary

Start by randomly generating a number X of the correct size. Then, randomly generate 101-bit factors p_1 and p_2 for $p \pm 1$. Using the Chinese Remainder Theorem (CRT) with p_1 and p_2, construct a sequence of candidates for p, starting at random point X, such that $p_1 \mid p-1$ and $p_2 \mid p+1$. Remove all candidates divisible by small primes with a sieve. Finally, test the candidates remaining after the sieve for primality using Miller-Rabin.

Procedure

Randomly select *and save* an integer X in the range

$$[\sqrt{2}\ 2^{511+128s},\ 2^{512+128s} - 1]$$

The prime p will be selected as the first integer greater than X which satisfies the strong prime requirements. The product, $n = pq$, of two of these randomly chosen primes will produce the public modulus which will have exactly 1024+256*s* bits.

Start by randomly generating two 101-bit numbers, y_1 and y_2. Using the sieve procedure outlined in Section *E.3* Sieving, generate a sequence of candidates for p_1 and p_2 by starting respectively at y_1 and y_2 and sieve out small primes. This will remove a substantial number of composite numbers that need not be tested for primality. Then test the survivors of the sieve for primality. The first two survivors satisfying primality shall be p_1 and p_2.

Starting at each of y_1 and y_2, sieve out all small primes up to $L=10^5$ over the range $[y_1, y_1 + 5 \times \log(p)]$, and $[y_2, y_2 + 5 \times \log(p)]$. The limit, 10^5, used to sieve the primes, is somewhat arbitrary, and is chosen for reasons of performance, rather than security. Any number between 10^3 and 10^6 is acceptable. Similarly, the length of the sieve interval, $M = 5 \times \log(p)$, is somewhat arbitrary. Numbers can be chosen which are convenient for the particular implementation of this procedure as dictated by resources such as the amount of computer memory available. The length of the sieve interval should be several times the size of the largest prime that is sieved. The numbers suggested are a good balance between the cost (in time) of the sieve procedure against the cost of testing candidates for primality. See Section E.3 Sieving for a full description of a sieve procedure.

The sieve will eliminate many of the numbers in the sieve interval. The numbers that are removed are divisible by small primes and hence are not candidates for primality testing. After sieving, test the remaining numbers in the sieve interval sequentially, starting at y_1 and y_2 to see if they are prime. Apply 27 iterations of Miller-Rabin to each candidate. This will result in a chance of error of less than 2^{-100}. These numbers can also be rigorously proven to be prime (if desired) by applying the Selfridge improvements to the theorem of Proth, Pocklington, & Lehmer [14]. This algorithm is fairly simple to implement and can *prove* primality of 101-bit numbers in well under a second on a modern PC. This procedure will yield two primes p_1 and p_2 which will be used as the large prime factors of p-1 and p+1 respectively.

NOTE
Using a factor base of all the primes up to 10^5 will remove approximately 95% of the composites in the sieve interval.

These 101-bit generated primes correspond to the *B1* limit discussed in [13]. It is well beyond computer range to apply the $p \pm 1$ algorithms up to $2^{100} \approx 10^{30}$. In fact, it can't be done within the lifetime of the universe with existing hardware.

At this point, the Chinese Remainder Theorem is used to construct a sequence of integers Y_i, starting at X such that every integer in the sequence is congruent to 1 mod p_1 and -1 mod p_2. This means that every integer Y_i in the sequence will have $p_1 \mid Y-1$ and $p_2 \mid Y+1$. Compute **R** as follows:

$$\mathbf{R} = ((p_2^{-1}) \bmod p_1)\, p_2 - ((p_1^{-1}) \bmod p_2)\, p_1.$$

If $\mathbf{R} < 0$, replace **R** by $\mathbf{R} + p_1 p_2$. Then compute Y_0 as follows:

$$Y_0 = X + (\mathbf{R} - X \bmod p_1\, p_2).$$

This is the first integer greater than X which is 1 mod p_1 and -1 mod p_2. Starting at Y_0, sieve the following integers:

$$Y_0,\ Y_0+p_1\, p_2,\ Y_0+2p_1\, p_2,\ Y_0+3p_1\, p_2,\ \ldots$$

by all small primes up to (say) 10^6. The value of 10^6 may be changed to anything that is convenient. The length of the sieve interval should be several times the largest prime in the sieve factor base (5 x 10^6 is a good choice). The integers untouched by the sieve will be candidates for primality testing and, as a result of our use of the CRT, will automatically be 'strong' primes. The public exponent e is sieved at this time, so that candidates p with $e \mid p-1$ are also removed.

After p has been computed, q is generated with the exact same procedure with one exception. When computing q, subtract the value of X computed for p from the value of X computed for q. If the absolute value of the difference is at least 2^{412}, then continue with the rest of the algorithm. If not, generate another X until the difference does exceed 2^{412}. The probability of having to generate more than one value of X for q is at most 2^{-100}.

E.5 Even Exponents

When **R** is constructed via the CRT, the conditions that **R** = 3 mod 8 (for the first prime to be generated) or **R** = 7 mod 8 (for the second prime to be generated) could be added in the case where an even integer is used as the public exponent, (i.e. the Rabin-Williams system). Therefore, the Rabin-Williams method can be chosen at essentially no computing cost, since the additional time for computing the CRT is negligible. The public exponent is also sieved, as in the odd exponent case. The algorithm for computing the CRT with more than two moduli can be found in [6, Algorithm 2.121]. When e is even, the integer sequence $\mathbf{Y}_i$ is:

$$\{\mathbf{Y}_0,\ \mathbf{Y}_1=\mathbf{Y}_0+(8p_1 p_2),\ \mathbf{Y}_2=\mathbf{Y}_0+2(8p_1 p_2),\ \mathbf{Y}_3=\mathbf{Y}_0+3(8p_1 p_2),\ \ldots\ ,\ \mathbf{Y}_j=\mathbf{Y}_0+j(8p_1 p_2)\ \}$$

E.6 Testing Candidates

Once the set of candidates has been sieved by small primes, the numbers surviving the sieve for primality can now be tested. There are several ways to do this. The set of possible methods has been greatly extended by new results which are discussed below. The basic criteria shall be that any method used must have an error rate no greater then $2^{-100} \approx 7.8 \times 10^{-31}$.

1. A deterministic primality test: The two best current methods for testing primality are the Cyclotomic ring test by Bosma-Cohen-Lenstra [8] or the Elliptic Curve Primality Test by Atkin-Goldwasser-Killian [7]. Although these are possibilities, they are not recommended, since these algorithms are quite complicated to implement and the likelihood of error in the implementation far exceeds the likelihood that a random method will return a composite.

2. Use of the Miller-Rabin algorithm: Sufficient tests are applied so that the probability that a randomly generated candidate is actually composite, when multiple Miller-Rabin tests indicate that the candidate is "prime", is less than 2^{-100}. According to [9, 18] for 512-bit primes, 8 iterations suffice. For 640-bit primes, 6 iterations suffice. **This is the recommended method for this standard**. Other possibilities for the testing of probabilistic primes are given below.

   Rather than generating so-called probabilistic primes, generation methods are available which result in numbers actually proven to be prime, by virtue of the constructive algorithms used. For an example of such a technique, refer to [20].

3. Use of the Miller-Rabin algorithm combined with a Lucas or Frobenius strong probable prime test[8]: According to Grantham, if a Frobenius probable prime test is used, the probability that a candidate is composite when the tests indicate that the candidate is "prime" is less than 1/1770, as opposed to the 1/4 probability obtained with Miller-Rabin alone. If a single Miller-Rabin test is performed, followed by T Frobenius tests, the probability of error for 512-bit primes is then less than $1.5 \times 10^{-17} (1/1770)^T$ [12, equation 1.6]. To achieve a probability of 2^{-100}, T=4 suffices. The analytical techniques of [9] should be able to be applied to the Grantham algorithm to arrive at even stronger probability estimates. That is to say, the number 1.5×10^{-17} can probably be reduced, but the method is as yet too new for sufficient analysis to have been performed. However, the correctness of the 1.5×10^{-17} bound is not in question.

4. It should be noted that there is no known composite integer which passes a single Miller-Rabin test, followed by a single Lucas strong probable prime test. While a formal estimate of the probability of error for a combined Miller-Rabin/Lucas test is still lacking, heuristics suggest that counter-examples are *extremely* rare. This combination of tests was suggested in [12]. The Miller-Rabin tests, followed by a single Lucas test is the recommended method for this standard.

   NOTE
   This subject area is changing. The purpose of the above discussions is simply to demonstrate that there are stronger alternatives to Miller-Rabin which can be used if desired.

[8] Jon Grantham, "*A Frobenius probable prime test with high confidence*", <http://www.math.uga.edu/~grantham/pseudo/pseudo2.ps>