PKI Algorithms

Centre for Development of Advanced Computing (C-DAC)
Bangalore

Under the Aegis of

Controller of Certifying Authorities (CCA)
Government of India

Overview

- Public key algorithms
  - Diffie-Hellman Key Exchange
  - RSA
- Hashing
  - SHA

Public Key Algorithms

- A class of cryptographic algorithms that requires two keys
  - One of which is secret, and another is public
- They are based on mathematical problems for which no efficient solutions are available
  - Integer Factorization
  - Discrete Logarithm
    - Integer k solving the equation $b^k = g$
  - Elliptic Curve Relationship

Diffie-Hellman Key Exchange

Modular Arithmetic

- If current time is 9 o'clock , after 4 hours the clock will show...
  - It is not at 13 o'clock , but it is 1o'clock.
  - When we reach 12, we start over; 12 is called the modulus.
- Example
  - 10 mod 6=4 5 mod 9 = 5
  - 32 mod 8 = 0

Diffie-Hellman Key Exchange

- Public key algorithm for key exchange
- Allows two users to exchange a secret key over an insecure medium without any prior secrets.
  - Functionality limited to key exchange only
- The scheme was first published by Whitfield Diffie and Martin Hellman in 1976.

Diffie-Hellman Key Exchange

Jai

Insecure Medium

Veeru

Select numbers

n, g and x

Calculate $Ak=g^x \bmod n$

Sending n, g and Ak

(n, g, Ak)

Select number y

Calculate $Bk=g^y \bmod n$

Sending Bk

Bk

Calculating Shared Key Ks

$Ks= Bk^{x}$

Calculating Shared Key Ks

$Ks= Ak^{y}$

Diffie-Hellman Key Exchange

$n=97$ $g=5$

Jai

Private Key $Xa=36$

Public Key $Ya=5^{36} \bmod 97$

$= 50 \bmod 97$

44

$44^{36} \bmod 97 = 75 \bmod 97$

Secret Key=75

Insecure Channel

97, 5, 50

44

Veeru

Private Key $Xb=58$

97,5, 50

Public Key $Yb=5^{58} \bmod 97$

$= 44 \bmod 97$

$50^{58} \bmod 97 = 75 \bmod 97$

Secret Key=75

Limitations of Diffie- Hellman Algorithm

- ☹Cannot be used for Encryption/Decryption and Digital Signature
- ☹Does not provide authentication to the communication parties.
  - ☹ Vulnerable to man-in the middle attack

RSA Algorithm

RSA Algorithm

- RSA
  - Developed by Ronald Rivest, Adi Shamir and Leonard Adleman
- Functionality
  - Encryption
  - Decryption
  - Digital signatures
- Based on number theory
  - Factoring a 232-digit number utilizing hundreds of machines took over 2 years to conclude

RSA Algorithm

- Key Generation Steps
  - Choose two large prime numbers p and q
  - Compute **n** and **z** such that **n=p*q** **,** **z=(p-1)*(q-1)**
  - Choose a number **d** relatively prime to **z**
  - Compute e such that (e*d)= 1 mod z

Public Key (n,e)

Private Key (n,d)

RSA Algorithm

- Encryption
  - Message m, cipher c
  - $C = m^{e} \bmod n$
  - where (n,e)receivers public key
- Decryption
  - $m = c^{d} \bmod n$

  where (n,d) receivers private key

RSA – Key Generation Example

- Select primes p=11, q=3.
- n = p*q = 11*3 = 33
- z = (p-1)(q-1) = 10*2 = 20
- Choose d=7
- Compute 'e' such that 7*e = 1 mod 20

  荇 e=3
- Public key = (n, e) = (33, 3)
- Private key = (n, d) = (33, 7).

RSA – Encryption and Decryption

- Encrypt the message m = 7

$c = m^e \bmod n = 7^3 \bmod 33 = 343 \bmod 33$

cipher text c = 13

- Decryption

$m' = c^d \bmod n = 13^7 \bmod 33 = 7$

Example

- An example of the RSA algorithm.

Plaintext (P)			Ciphertext (C)		After decryption	
Symbolic	Numeric	P^3	P^3 (mod 33)	C^7	C^7 (mod 33)	Symbolic
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	05	E

Sender's computation

Receiver's computation

ASN.1 STRUCTURE OF RSA PUBLIC AND PRIVATE KEYS IN DER FORMAT

RSA Private Key


```
-----BEGIN RSA PRIVATE KEY-----
30 82 02 5e 02 01 00 02 81 81 00 c7 8a 46 ac b5 cf 3e 23 cd 73 0a
0f ea 59 6f 1d 32 bf 26 aa f5 d0 de a6 cf 02 6b b4 46 3c 68 65 52
38 ee db ec 91 89 45 01 73 9f d2 c3 eb 84 ed a7 52 ea 28 26 78 27
1d 5d 3a df d8 93 4c 46 06 d6 f7 24 35 a5 b6 47 a5 39 41 37 50 e5
1a b9 bb eb 95 de 93 24 ef 0e d5 b3 89 f7 ba b4 3a 8e 7a ad da b4
d7 6c 2d 43 35 af cf 15 e0 19 6a d6 df ed f7 c1 07 2d 08 18 ed 33
73 5d bc 22 c8 58 73 02 03 01 00 01 02 81 81 00 b4 cd 97 62 71 4e
fa b8 48 35 bf dd 51 f4 7d 99 10 5d 61 f5 30 cd 74 a1 e3 1b 07 6a
8e e5 b7 96 6f 5d 45 19 a3 8e ef b9 c6 29 f5 9c 6d 88 1f a7 93 a0
ae a9 78 ca 10 6f 2c 05 e7 c4 7f 1b 72 aa b0 39 1b 45 86 5c 95 eb
c5 35 cb 84 7e f1 6c 43 36 1d 1a 7a fc 47 f2 14 52 ff 55 f9 d3 7d
43 26 6d 49 95 4d 04 fa f4 ad 8a 4b 4d d2 d0 b7 79 b1 f9 84 4f b5
3b 16 c8 1d 26 7e 58 e5 8b 61 7e c9 02 41 00 e6 e8 a0 46 02 a9 f1
27 06 6a e5 22 8e f7 4e 2d 92 21 1d 6a bb cb 95 b3 fa 73 87 52 e0
f0 f0 fe f0 39 32 94 df 84 09 02 85 84 7b be f5 a7 d3 df fc 64 c0
fe 4a f4 05 09 cf 42 f5 94 0f 11 b3 27 02 41 00 dd 39 0b a9 2b c0
f7 ab c8 52 ab 2a d8 ce 18 92 27 88 8a 18 ef 6c 11 9e 83 5c 74 41
2c dc 70 d1 85 fe 3c 61 61 68 ae 24 5b 67 29 73 cf 33 45 cb f8 26
cd 3e 97 25 04 df 92 d1 aa ab e1 59 0f d5 02 40 62 09 68 6a f1 1c
98 1f 9a 90 ee 02 13 33 1a c5 2c 62 d4 eb 89 1a 31 d0 3d 48 a9 ae
dd 84 4e bd de de 5a 04 6d 35 d0 e1 7a 30 ba 9d 64 0d 42 5e b9 f9
49 1b 6e 55 56 82 48 b6 44 2f fd 89 e5 47 02 41 00 c0 77 73 17 b9
c3 67 37 83 4f b9 2f cb f4 73 18 25 60 fb 94 fa 28 b1 a3 91 72 0c
8a ef b6 d2 48 d8 24 fa ef 56 4a 36 c7 d6 e6 08 00 83 d2 7d f5 19
6e d8 be 8d cd 5d 52 0e 70 6f e6 9e 66 58 09 02 41 00 ba ba e9 9a
9a a3 e6 f8 34 f3 b6 67 80 a7 c5 19 1e 6b a3 30 e2 4e c2 b9 0d c7
93 57 de ca 6b 76 3e 37 39 7e 5d 64 f1 0c 15 a4 e1 83 e1 d9 99 4a
bf 8d 36 85 91 1f 58 16 31 50 39 f9 66 41 ff 6f
-----END RSA PRIVATE KEY-----
```

RSA Private Key

Within the RSA, PKCS#1 uses the **Distinguished Encoding Rules (DER)** encoding of ASN.1 to represent keys in a portable format.

The ASN.1 private key structure in DER format has the following components:

```
RSAPrivateKey :: = SEQUENCE {
        version             Version,
        modulus             INTEGER ,   n
        publicExponent      INTEGER ,   e
        privateExponent     INTEGER ,   d
        prime1              INTEGER ,   p
        prime2              INTEGER ,   q
        exponent1           INTEGER ,   d mod (p-1)
        exponent2           INTEGER ,   d mod (q-1)
        coefficient         INTEGER ,  (inverse of q) mod p
        otherPrimeInfos     OPTIONAL
       }
```

Extracted Private Key Components

Header: 30 82 02 5e (0x3082 = = ASN.1 Sequence)

Separator: 02 01 (0x02 = = Integer and 0x01 = = 1 byte long)

Algorithm Version: 00 (0x00 = = version zero)

Separator: 02 81 81 (0x02 = = Integer and 0x81 = = 129 byte long)

Modulus: (129 bytes- starts with a null (0x00) , remove this)

00	c7	8a	46	ac	b5	cf	3e	23	cd	73	0a	0f	ea	59
6f	1d	32	bf	26	aa	f5	d0	de	a6	cf	02	6b	b4	46
3c	68	65	52	38	ee	db	ec	91	89	45	01	73	9f	d2
c3	eb	84	ed	a7	52	ea	28	26	78	27	1d	5d	3a	df
d8	93	4c	46	06	d6	f7	24	35	a5	b6	47	a5	39	41
37	50	e5	1a	b9	bb	eb	95	de	93	24	ef	0e	d5	b3
89	f7	ba	b4	3a	8e	7a	ad	da	b4	d7	6c	2d	43	35
af	cf	15	e0	19	6a	d6	df	ed	f7	c1	07	2d	08	18
ed	33	73	5d	bc	22	c8	58	73						

Separator: 02 03 (0x02 = = Integer and 0x03 = = 3 byte long)

CCA

Extracted Private Key Components

रसी डैक CDAC

Public Exponent: 01 00 01 (Integer value 65537, Fermat number F4 , 3bytes)

Separator: 02 81 81 (0x02 = = Integer and 0x81 = = 129 byte long)

Private Component: (129 bytes- starts with a null , remove this)

00	b4	cd	97	62	71	4e	fa	b8	48	35	bf	dd	51	f4
7d	99	10	5d	61	f5	30	cd	74	a1	e3	1b	07	6a	8e
e5	b7	96	6f	5d	45	19	a3	8e	ef	b9	c6	29	f5	9c
6d	88	1f	a7	93	a0	ae	a9	78	ca	10	6f	2c	05	e7
c4	7f	1b	72	aa	b0	39	1b	45	86	5c	95	eb	c5	35
cb	84	7e	f1	6c	43	36	1d	1a	7a	fc	47	f2	14	52
ff	55	f9	d3	7d	43	26	6d	49	95	4d	04	fa	f4	ad
8a	4b	4d	d2	d0	b7	79	b1	f9	84	4f	b5	3b	16	c8
1d	26	7e	58	e5	8b	61	7e	c9						

Separator: 02 41 (0x02 = = Integer and 0x41 = = 65 byte long)

Extracted Private Key Components

Prime 1: (65 bytes- starts with a null)

```
00 e6 e8 a0 46 02 a9 f1 27 06 6a e5 22 8e f7
4e 2d 92 21 1d 6a bb cb 95 b3 fa 73 87 52 e0
f0 f0 fe f0 39 32 94 df 84 09 02 85 84 7b be
f5 a7 d3 df fc 64 c0 fe 4a f4 05 09 cf 42 f5
94 0f 11 b3 27
```

Separator: 02 41 (0x02 = = Integer and 0x41 = = 65 byte long)

Prime 2: (65 bytes- starts with a null)

```
00 dd 39 0b a9 2b c0 f7 ab c8 52 ab 2a d8 ce
18 92 27 88 8a 18 ef 6c 11 9e 83 5c 74 41 2c
dc 70 d1 85 fe 3c 61 61 68 ae 24 5b 67 29 73
cf 33 45 cb f8 26 cd 3e 97 25 04 df 92 d1 aa
ab e1 59 0f d5
```

Separator: 02 40 (0x02 = = Integer and 0x40 = = 64 byte long)

Extracted Private Key Components

Exponent 1: (64 bytes)

```
62 09 68 6a f1 1c 98 1f 9a 90 ee 02 13 33 1a
c5 2c 62 d4 eb 89 1a 31 d0 3d 48 a9 ae dd 84
4e bd de de 5a 04 6d 35 d0 e1 7a 30 ba 9d 64
0d 42 5e b9 f9 49 1b 6e 55 56 82 48 b6 44 2f
fd 89 e5 47
```

Separator: 02 41 (0x02 = = Integer and 0x41 = = 65 byte long)

Exponent 2: (64 bytes)

```
00 c0 77 73 17 b9 c3 67 37 83 4f b9 2f cb f4
73 18 25 60 fb 94 fa 28 b1 a3 91 72 0c 8a ef
b6 d2 48 d8 24 fa ef 56 4a 36 c7 d6 e6 08 00
83 d2 7d f5 19 6e d8 be 8d cd 5d 52 0e 70 6f
e6 9e 66 58 09
```

Separator: 02 41 (0x02 = = Integer and 0x41 = = 65 byte long)

Extracted Private Key Components

Coefficient: (65 bytes, always starts with a null)

```
00 ba ba e9 9a 9a a3 e6 f8 34 f3 b6 67 80 a7
c5 19 1e 6b a3 30 e2 4e c2 b9 0d c7 93 57 de
ca 6b 76 3e 37 39 7e 5d 64 f1 0c 15 a4 e1 83
e1 d9 99 4a bf 8d 36 85 91 1f 58 16 31 50 39
f9 66 41 ff 6f
```

RSA Private Key

HEADER ALGORITHM VERSION PUBLIC EXPONENT

-----BEGIN RSA PRIVATE KEY-----

	30	82	02	5e	02	01	00	02	81	81	00	c7	8a	46	ac	b5	cf	3e	23	cd	73	0a
	[illegible]	[illegible]	[illegible]	[illegible]	1d	32	bf	26	aa	f5	d0	de	a6	cf	02	6b	b4	46	3c	68	65	52
	38	ee	db	ec	91	89	45	01	73	9f	d2	c3	eb	84	ed	a7	52	ea	28	26	78	27
MODULUS	1d	5d	3a	df	d8	93	4c	46	06	d6	f7	24	35	a5	b6	47	a5	39	41	37	50	e5
	1a	b9	bb	eb	95	de	93	24	ef	0e	d5	b3	89	f7	ba	b4	3a	8e	7a	ad	da	b4
	d7	6c	2d	43	35	af	cf	15	e0	[illegible]	6a	[illegible]	df	ed	f7	c1	07	2d	08	18	ed	33
	73	5d	bc	22	c8	58	73	02	03	01	00	01	02	81	81	00	b4	cd	97	62	71	4e
	fa	b8	48	35	bf	dd	51	f4	7d	[illegible]	10	[illegible]	61	f5	30	cd	74	a1	e3	1b	07	6a
	8e	e5	b7	96	6f	5d	45	19	a3	8e	ef	b9	c6	29	f5	9c	6d	88	1f	a7	93	a0
PRIVATE	ae	a9	78	ca	10	6f	2c	05	e7	c4	7f	1b	72	aa	b0	39	1b	45	86	5c	95	eb
COMPONENT	c5	35	cb	84	7e	f1	6c	43	36	1d	1a	7a	fc	47	f2	14	52	ff	55	f9	d3	7d
	43	26	6d	49	95	4d	04	fa	f4	ad	8a	4b	4d	d2	d0	b7	79	b1	f9	84	4f	b5
	3b	16	c8	1d	26	7e	58	e5	8b	61	7e	c9	02	41	00	e6	e8	a0	46	02	a9	f1
	27	06	6a	e5	22	8e	f7	4e	2d	92	21	1d	6a	bb	cb	95	b3	fa	73	87	52	e0
PRIME 1	f0	f0	fe	f0	39	32	94	df	84	09	02	85	84	7b	be	f5	a7	d3	df	fc	64	c0
	fe	4a	f4	05	09	cf	42	f5	94	0f	11	b3	27	02	41	00	dd	39	0b	a9	2b	c0
	f7	ab	c8	52	ab	2a	d8	ce	18	92	27	88	8a	18	ef	6c	11	9e	83	5c	74	41
PRIME 2	2c	dc	70	d1	85	fe	3c	61	61	68	ae	24	5b	67	29	73	cf	33	45	cb	f8	26
	cd	3e	97	25	04	df	92	d1	aa	ab	e1	59	0f	d5	02	40	62	09	68	6a	f1	1c
	98	1f	9a	90	ee	02	13	33	1a	c5	2c	62	d4	eb	89	1a	31	d0	3d	48	a9	ae
EXPONENT 1	dd	84	4e	bd	de	de	5a	04	6d	35	d0	e1	7a	30	ba	9d	64	0d	42	5e	b9	f9
	49	1b	6e	55	56	82	48	b6	44	2f	fd	89	e5	47	02	41	00	c0	77	73	17	b9
	c3	67	37	83	4f	b9	2f	cb	f4	73	18	25	60	fb	94	fa	28	b1	a3	91	72	0c
EXPONENT 2	8a	ef	b6	d2	48	d8	24	fa	ef	56	4a	36	c7	d6	e6	08	00	83	d2	7d	f5	19
	6e	d8	be	8d	cd	5d	52	0e	70	6f	e6	9e	66	58	[illegible]	02	41	[illegible]	ba	ba	e9	9a
	9a	a3	e6	f8	34	f3	b6	67	80	a7	c5	19	1e	6b	a3	[illegible]	e2	4e	c2	b9	0d	c7
COEFFICIENT	93	57	de	ca	6b	76	3e	37	39	7e	5d	64	f1	0c	15	a4	e1	83	e1	d9	99	4a
	bf	8d	36	85	91	1f	58	16	31	50	39	f9	66	41	ff	6f						

-----END RSA PRIVATE KEY-----

SEPERATOR

RSA Public Key


```
-----BEGIN PUBLIC KEY-----
30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 81 8d 00
30 81 89 02 81 81 00 c7 8a 46 ac b5 cf 3e 23 cd 73 0a 0f ea 59 6f
1d 32 bf 26 aa f5 d0 de a6 cf 02 6b b4 46 3c 68 65 52 38 ee db ec
91 89 45 01 73 9f d2 c3 eb 84 ed a7 52 ea 28 26 78 27 1d 5d 3a df
d8 93 4c 46 06 d6 f7 24 35 a5 b6 47 a5 39 41 37 50 e5 1a b9 bb eb
95 de 93 24 ef 0e d5 b3 89 f7 ba b4 3a 8e 7a ad da b4 d7 6c 2d 43
35 af cf 15 e0 19 6a d6 df ed f7 c1 07 2d 08 18 ed 33 73 5d bc 22
c8 58 73 02 03 01 00 01
-----END PUBLIC KEY-----
```

RSA Public Key

The ASN.1 public key structure in DER format has the following components:

```
RSAPublicKey  :: =  SEQUENCE {
                        modulus              INTEGER ,  n
                        publicExponent       INTEGER ,   e
                      }
```

Extracted Public Key Components

Header:

```
30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d 01 01
01 05 00 03 81 8d 00 30 81 89
```

Separator: 02 81 81 (0x02 = = Integer and 0x81 = = 129 byte long)

Modulus:

```
00 c7 8a 46 ac b5 cf 3e 23 cd 73 0a 0f ea 59
6f 1d 32 bf 26 aa f5 d0 de a6 cf 02 6b b4 46
3c 68 65 52 38 ee db ec 91 89 45 01 73 9f d2
c3 eb 84 ed a7 52 ea 28 26 78 27 1d 5d 3a df
d8 93 4c 46 06 d6 f7 24 35 a5 b6 47 a5 39 41
37 50 e5 1a b9 bb eb 95 de 93 24 ef 0e d5 b3
89 f7 ba b4 3a 8e 7a ad da b4 d7 6c 2d 43 35
af cf 15 e0 19 6a d6 df ed f7 c1 07 2d 08 18
ed 33 73 5d bc 22 c8 58 73
```

Extracted Public Key Components

Separator: 02 03 (0x02 = = Integer and 0x03 = = 3 bytes long)

Public Exponent: 01 00 01

RSA Public Key

SEPERATOR

HEADER

MODULUS

PUBLIC EXPONENT

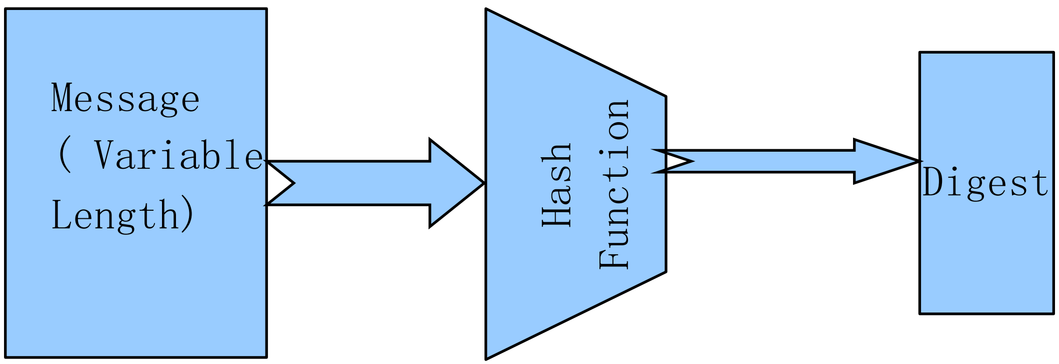
```
-----BEGIN PUBLIC KEY-----
30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 81 8d 00
30 81 89 02 81 81 00 c7 8a 46 ac b5 cf 3e 23 cd 73 0a 0f ea 59 6f
1d 32 bf 26 aa f5 d0 de a6 cf 02 6b b4 46 3c 68 65 52 38 ee db ec
91 89 45 01 73 9f d2 c3 eb 84 ed a7 52 ea 28 26 78 27 1d 5d 3a df
d8 93 4c 46 06 d6 f7 24 35 a5 b6 47 a5 39 41 37 50 e5 1a b9 bb eb
95 de 93 24 ef 0e d5 b3 89 f7 ba b4 3a 8e 7a ad da b4 d7 6c 2d 43
35 af cf 15 e0 19 6a d6 df ed f7 c1 07 2d 08 18 ed 33 73 5d bc 22
c8 58 73 02 03 01 00 01
-----END PUBLIC KEY-----
```

Cryptography Hash Algorithm

Hash Function

- A hash function is an algorithm
  - which creates a digital representation or "fingerprint" in the form of a "hash value" or "hash result" of a standard length
  - Hash value is usually much smaller than the message but nevertheless substantially unique to it.
  - Any change to the message invariably produces a different hash result when the same hash function is used
  - It is a one way function – Easy to calculate hash value from message but difficult to generate message from hash value

Hash Function



Secure Hash Algorithm (SHA)

- SHA was designed by NIST & NSA in 1993, revised 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - the algorithm is SHA, the standard is SHS
- produces 160-bit hash values
- now the generally preferred hash algorithm

SHA - steps

1. Pad message so its length is 448 mod 512
2. Append a 64-bit length value to message
3. Initialize 5-word (160-bit) buffer (A,B,C,D,E) to

   (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. Process message in 16-word (512-bit) chunks:
   - expand 16 words into 80 words by mixing & shifting
   - use 4 rounds of 20 bit operations on message block & buffer
   - add output to input to form new buffer value
5. Output hash value is the final buffer value

SHA Vs MD5

- SHA
  - Brute force attack is harder (160 Vs 128 bits for MD5)
  - Not vulnerable to any known cryptanalytic attacks (compared to MD4/5)
  - A little slower than MD5 (80 Vs 64 steps)
- Both work well on a 32-bit architecture
- Both designed as simple and compact for implementation

Elliptic Curve Cryptography

- Based on Algebraic structure of elliptic curves over finite fields
- Elliptic Curves can be applied for
  - Digital Signatures
  - Encryption
  - Pseudo-random Generators
- Use of Elliptic curves in cryptography was suggested by Neal Koblitz and Victor Miller in 1985
  - Widely adopted in 2004-2005

Advantages of ECC

- Smaller Key Size
  - Gives equivalent security strength of traditional crypto systems
- Reduced Computational Power

ECC Guidelines issued in 2013 for public scrutiny

Security Strength (symmetric key)	RSA key size	Equivalent ECC Key Size
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Curves	EC Key Size	Message digest Algorithms
P-224	224	SHA-256
P-256	256	SHA-256
P-384	384	SHA-384
P-521	512	SHA-512

References

- Cryptography and Network security – principles and practice : William Stallings
- Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C : Bruce Schneier
- www.certicom.com/index.php/ecc-turorial
- http://campustechnology.com/articles/39190_2
- http://csrc.nist.gov/
- http://www-fs.informatik.uni-tuebingen.de/~reinhard/krypto/English/english.html

Thank You

pki@cdac.in