

High-Speed Cryptography

Çetin K. Koç

Oregon State University

1

Contents

- Intel Cryptographic Library
- Optimization of RSA & DSS
- Exponentiation & Multiplication
- Montgomery Multiplication
- Implementation Results
- ◇ High-Speed Software for Crypto
- ◇ Cryptographer's Wish List
- ◇ MMX Implementation
- ◇ Elliptic Curve Cryptography

2

Intel Cryptographic Library

Crypto Service Provider (CSP) within CDSA

- RSA Sign, Verify, & Key Generate
- DSS Sign, Verify, & Key Generate
- Diffie-Hellman Key Agreement
- SHA, MD2, & MD5
- DES, 3DES, RC2, RC4, & RC5
- Random Number Generator
- Password Based Encryption

3

RSA Algorithm

The RSA algorithm uses modular exponentiation

$$C = M^e \pmod{n}$$

for obtaining and verifying digital signatures

The computation of $M^e \pmod{n}$ is performed using exponentiation algorithms (heuristics)

Modular exponentiation requires implementation of 3 basic modular arithmetic operations: **addition**, **subtraction**, and **multiplication**

The signature operation can be decomposed into two half-size modular exponentiations using the Chinese remainder theorem

4

DSS Algorithm

p is prime
 q is prime (dividing $p - 1$)
 g is the q th root of 1 modulo p
 x is private key (secret integer less than $q - 1$)
 $y = g^x \pmod{p}$ is the public key

The signature on M and k is the pair (r, s)

$$\begin{aligned}
 r &:= (g^k \pmod{p}) \pmod{q} \\
 s &:= (M + xr)k^{-1} \pmod{q}
 \end{aligned}$$

The signature verification

$$\begin{aligned}
 w &:= s^{-1} \pmod{q} \\
 u_1 &:= Mw \pmod{q} \\
 u_2 &:= rw \pmod{q} \\
 v &:= (g^{u_1}y^{u_2} \pmod{p}) \pmod{q}
 \end{aligned}$$

Check if $r = v$

5

Exponentiation Heuristics

Given the integer e , the computation of M^e is an exponentiation operation

The objective is to use as few multiplications as possible for a given integer e

This problem is related to **addition chains**

An addition chain is a sequence of integers

$$\begin{array}{ccccccc}
 a_0 & a_1 & a_2 & \cdots & a_r \\
 1 & & & & e
 \end{array}$$

from 1 to e such that any a_k is the sum of two earlier integers a_i and a_j

$$a_k = a_i + a_j \quad \text{for } 0 < i, j < k$$

6

Addition Chain Example

$e = 55$

```

1 2 3 6 12 13 26 27 54 55
1 2 3 6 12 13 26 52 55
1 2 4 5 10 20 40 50 55
1 2 3 5 10 11 22 44 55
    
```

An addition chain yields an algorithm for computing M^e given the integer e

$$M^1 M^2 M^3 M^5 M^{10} M^{11} M^{22} M^{44} M^{55}$$

The length of the chain r gives the number of operations required to compute M^e

7

Addition Chain Theory

Finding the shortest addition chain for a given e is an NP-complete problem

Upper bound: $\lfloor \log_2 e \rfloor + H(e) - 1$

Lower bound: $\log_2 e + \log_2 H(e) - 2.13$

Heuristics: binary, m -ary, sliding windows

Statistical methods, such as simulated annealing, can be used to produce short addition chains for certain exponents

8

The Binary Method

Scan the bits of e 1 bit at a time, and perform 1 squaring & 1 multiplication:

```
function  $C := M^e \pmod n$ 
1.  if  $e_{k-1} = 1$  then  $C := M$  else  $C := 1$ 
2.  for  $i = k - 2$  downto 0
2a.   $C := C \cdot C \pmod n$ 
2b.  if  $e_i = 1$  then  $C := C \cdot M \pmod n$ 
3.  return  $C$ 
```

The m -ary Method

Scan the bits of e w bits at a time, and perform w squarings and 1 multiplication with a value

$$\{M^0, M^1, M^2, \dots, M^{2^w-1}\}$$

in a precomputed table, where $w = \log_2 m$

9

Modular Multiplication

Given $A, B < n$, compute $P = A \cdot B \pmod n$

Methods:

- Multiply and reduce:
Multiply: $P' = A \cdot B$ ($2k$ -bit number)
Reduce: $P = P' \pmod n$ (k -bit number)
These steps can be interleaved
- Montgomery's method
Division by 2^k instead of division by n

10

Montgomery Multiplication

Assuming $\gcd(n, r) = 1$, we map the set

$$\{0, 1, 2, \dots, n-1\}$$

to itself using the one-to-one mapping

$$\bar{a} = a \cdot r \pmod n$$

The **Montgomery product** of two numbers is defined as

$$\text{MP}(\bar{a}, \bar{b}) = \bar{a} \cdot \bar{b} \cdot r^{-1} \pmod n$$

where r^{-1} is the inverse of $r \pmod n$

function $\text{MP}(\bar{a}, \bar{b})$

```
1.   $t := \bar{a} \cdot \bar{b}$ 
2.   $u := (t + (t \cdot n' \pmod r) \cdot n) / r$ 
3.  if  $u \geq n$  then  $u := u - n$ 
4.  return  $u$ 
```

Only mod r arithmetic is required

11

Montgomery Multiplication

- In order to compute $\text{MP}(\bar{a}, \bar{b})$, we need n'

$$r \cdot r^{-1} - n \cdot n' = 1$$

(the extended Euclidean algorithm)

- If $c = a \cdot b \pmod n$, then $\bar{c} = \text{MP}(\bar{a}, \bar{b})$

$$\begin{aligned} \bar{c} &= a \cdot b \cdot r^{-1} \pmod n \\ &= (a \cdot r) \cdot (b \cdot r) \cdot r^{-1} \pmod n \\ &= \text{MP}(\bar{a}, \bar{b}) \end{aligned}$$

- If \bar{c} is given, then $c = \text{MP}(\bar{c}, 1)$

$$\begin{aligned} c &= (c \cdot r) \cdot 1 \cdot r^{-1} \pmod n \\ &= \text{MP}(\bar{c}, 1) \end{aligned}$$

12

Montgomery Exponentiation

function $C := M^e \bmod n$

1. Compute n' using Euclidean algorithm
2. $\overline{M} := M \cdot r \bmod n$
3. $\overline{C} := 1 \cdot r \bmod n$
4. **for** $i = k - 1$ **downto** 0
 - 4a. $\overline{C} := \text{MP}(\overline{C}, \overline{C})$
 - 4b. **if** $e_i = 1$ **then** $\overline{C} := \text{MP}(\overline{C}, \overline{M})$
5. $C := \text{MP}(\overline{C}, 1)$
6. **return** C

13

Montgomery Algorithms

Separated Operand Scanning (SOS)

First computes $t = a \cdot b$ and then interleaves the computations of $m = t \cdot n' \bmod r$ and $u = (t + m \cdot n)/r$. Squaring can be optimized.

SOS requires $2s + 2$ words of space

Finely Integrated Product Scanning (FIPS)

Interleaves computation of $a \cdot b$ and $m \cdot n$ by scanning the words of m

It uses the same space to keep m and u , reducing the temporary space to $s + 3$ words

Coarsely Integrated Hybrid Scanning (CIHS)

The computation of $a \cdot b$ is split into 2 loops, and the second loop is interleaved with the computation of $m \cdot n$

CIHS also requires $s + 3$ words of space

14

Montgomery Algorithms

Finely Integrated Operand Scanning (FIOS)

The computation of $a \cdot b$ and $m \cdot n$ is performed in a single loop

FIOS also requires $s + 3$ words of space

Coarsely Integrated Operand Scanning (CIOS)

Improves the SOS method by integrating the multiplication and reduction steps. Instead of computing the entire product $a \cdot b$, then reducing, we alternate between iterations of the outer loops for multiplication and reduction

CIOS also requires $s + 3$ words of space

	Add	Read/Write	Space
SOS	$4s^2 + 4s + 2$	$8s^2 + 13s + 5$	$2s + 2$
FIPS	$6s^2 + 2s + 2$	$14s^2 + 16s + 3$	$s + 3$
CIHS	$4s^2 + 4s + 2$	$9.5s^2 + 11.5s + 3$	$s + 3$
FIOS	$5s^2 + 3s + 2$	$10s^2 + 9s + 3$	$s + 3$
CIOS	$4s^2 + 4s + 2$	$8s^2 + 12s + 3$	$s + 3$

All methods require $2s^2 + s$ multiplications

15

Derivation of CIOS

The binary Montgomery algorithm:

$$\begin{aligned}
 u &= \text{MP}(a, b) \\
 &= a \cdot b \cdot r^{-1} \pmod{n} \\
 &= a \cdot b \cdot 2^{-k} \pmod{n} \\
 &= \left(\sum_{i=0}^{k-1} a_i 2^i \right) \cdot b \cdot 2^{-k} \pmod{n} \\
 &= \left(\sum_{i=0}^{k-1} a_i 2^{i-k} \right) \cdot b \pmod{n}
 \end{aligned}$$

$$u = (a_0 2^{-k} + a_1 2^{-k+1} + \dots + a_{k-1} 2^{-1}) \cdot b \bmod n$$

1. $u := 0$
2. **for** $i = 0$ **to** $k - 1$
 - 2a. $u := u + a_i b$
 - 2b. **if** u is odd **then** $u := u + n$
3. $u := u/2$

16

Derivation of CIOS

- Compute $u := u + a_i \cdot b$
- Add an integer (m) multiple of n to u so that the LSB of $u + m \cdot n$ is zero

We have

$$u + m \cdot n = u_0 + m \cdot n_0 = 0 \pmod{2}$$

and thus,

$$m := u_0 \cdot (-n_0^{-1}) \pmod{2}$$

n is odd, and therefore $(-n_0^{-1}) = 1 \pmod{2}$

-
1. $u := 0$
 2. **for** $i = 0$ **to** $k - 1$
 - 2a. $u := u + a_i \cdot b$
 - 2b. $m := u_0 \cdot (-n_0^{-1}) \pmod{2}$
 - 2c. $u := u + m \cdot n$
 3. $u := u/2$
-

17

Derivation of CIOS

$k = sw$, where w is the wordsize

-
1. $u := 0$
 2. **for** $i = 0$ **to** $s - 1$
 - 2a. $u := u + a_i \cdot b$
 - 2b. $m := u_0 \cdot (-n_0^{-1}) \pmod{2^w}$
 - 2c. $u := u + m \cdot n$
 3. $u := u/2^w$
-

Since $r \cdot r^{-1} - n' \cdot n = 1$ and $r = 2^k$, we have

$$\begin{aligned} 2^{sw} \cdot 2^{sw} - n' \cdot n &= 1 \pmod{2^w} \\ -n'_0 \cdot n_0 &= 1 \pmod{2^w} \end{aligned}$$

therefore $(-n_0^{-1}) = n'_0$

18

Implementation Results

Pentium 120 MHz, Windows NT 3.51

Algorithm	C	Assembler
RSA Sign	306 msec	110 msec
RSA Verify	46 msec	15 msec
RSA KeyGen	19.95 sec	7.76 sec
DSS Sign	65 msec	40 msec
DSS Verify	215 msec	97 msec
DSS KeyGen	124.89 sec	48.12 sec
MD5	8.90 mb/s	15.10 mb/s
SHA	3.67 mb/s	5.77 mb/s
RC4	4.93 mb/s	7.97 mb/s
RC5	2.73 mb/s	6.26 mb/s
DES	1.32 mb/s	2.07 mb/s

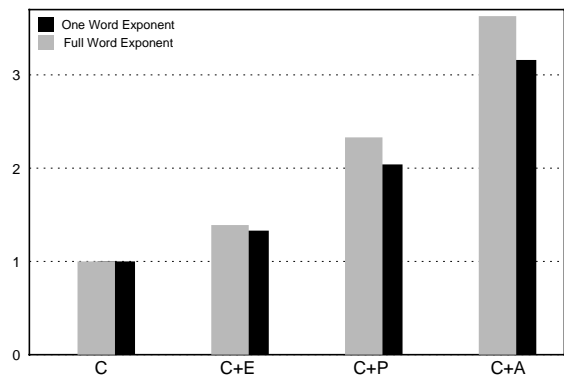
RSA and DSS: 1024 bits

19

High-Speed Software for Crypto

Compiler A vs Compiler B
 Compiler vs Assembler
 Cost vs Speed

- ANSI C implementation
- C + Extended Types (`_int64`, `long long`)
- C + Primitive Operations in Assembler
- C + Basic Functions in Assembler



20

Cryptographer's Wish List

Single instruction cycle for integer and $GF(2^k)$ primitive operations

Primitive Operations

ADD(C,S,a,b)	$(C, S) = a + b$
ADD2(C,S,a,b,c)	$(C, S) = a + b + c$
MUL(C,S,a,b)	$(C, S) = a \cdot b$
MUL2ADD(C,S,a,b,c,d)	$(C, S) = a \cdot b + c \cdot d$
MULADD(C,S,a,b,c)	$(C, S) = a \cdot b + c$
MULADD2(C,S,a,b,c,d)	$(C, S) = a \cdot b + c + d$
SQU(C,S,a)	$(C, S) = a^2$
SQUADD(C,S,a,b)	$(C, S) = a^2 + b$

21

Cryptographer's Wish List

$GF(2^k)$ primitive operations:

$$(C(x), S(x)) := S(x) + P(x) * Q(x) + C(x)$$

$A(x), B(x), C(x), S(x)$: degree 31 polynomials with coefficients from the field $GF(2)$

$$\begin{aligned} P(x) &= x^3 + x^2 = 1100 \\ Q(x) &= x^3 + x^2 + 1 = 1101 \\ P(x) * Q(x) &= (x^3 + x^2) * (x^3 + x^2 + 1) \\ &= x^6 + x^4 + x^3 + x^2 \\ &= 1011100 \end{aligned}$$

$$\begin{array}{r} \begin{array}{cccccc} 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ & & & & 1 & 1 & 0 & 0 \\ & & & & 1 & 1 & 0 & 1 \\ \hline & & & & 1 & 1 & 0 & 0 \\ & & & 0 & 0 & 0 & 0 & \\ & & 1 & 1 & 0 & 0 & & \\ 1 & 1 & 0 & 0 & & & & \\ \hline 1 & 0 & 1 & 1 & 1 & 0 & 0 & \end{array} \end{array}$$

22

MMX Implementation

MMX Enhancements

- Register Usage
- Instruction Usage
- Selection of Algorithm

SOS, FIPS, CIHS, **FIOS**, CIOS

for $j = 0$ to $s - 1$
 $(C, S) := t_j + a_j \cdot b_i + C$
 ADD (t_{j+1}, C)
 $(C, S) := S + m \cdot n_j$
 $t_{j-1} := S$

23

FIOS on MMX

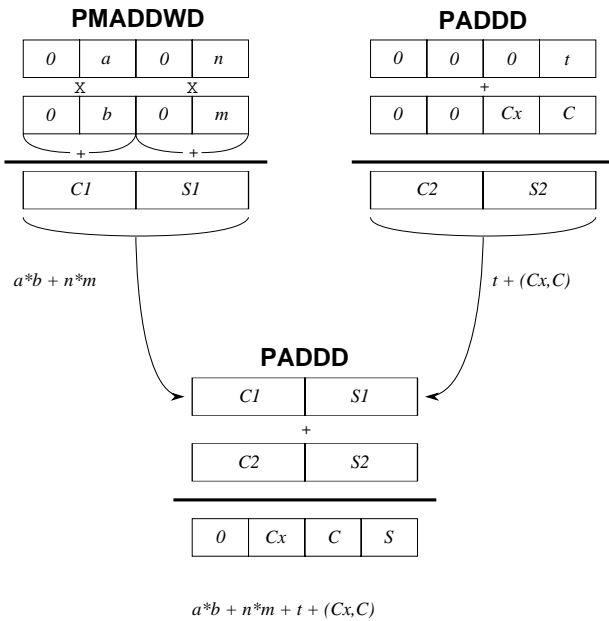
FIOS Pseudocode for MMX:

for $j = 0$ to $s - 1$
 $(Cx, C, S) := t_j + a_j \cdot b_i + m \cdot n_j + C + Cx \cdot 2^w$
 $t_{j-1} := S$

Operation	Instruction
$(C1, S1) := a \cdot b + m \cdot n$	PMADDWD
$(C2, S2) := (Cx, C) + t$	PADDD
$(C, S) := (C1, S1) + (C2, S2)$	PADDD

24

MMX Instruction Usage



25

Elliptic Curve Cryptosystems

Elliptic curves defined over $GF(p)$ or $GF(2^k)$ are used in cryptography

The arithmetic of $GF(p)$ is the usual mod p arithmetic

The arithmetic of $GF(2^k)$ is similar to that of $GF(p)$, however, there are some differences

Elliptic curves over $GF(2^k)$ are more popular due to the space and time-efficient algorithms for doing arithmetic in $GF(2^k)$

Elliptic curve cryptosystems based on discrete logarithms seem to provide similar amount of security to that of RSA, but with relatively shorter key sizes

26

Elliptic Curves over $GF(2^k)$

A non-supersingular elliptic curve E over the field $GF(2^k)$ is defined by parameters $a, b \in GF(2^k)$ with $b \neq 0$ is the set of solutions (x, y) where $x, y \in GF(2^k)$, to the equation

$$y^2 + xy = x^3 + ax^2 + b$$

together with an extra point O . The set of points E form a group with respect to the addition rules:

- $O + O = O$
- $(x, y) + O = (x, y)$
- $(x, y) + (x, x + y) = O$

27

Elliptic Curves over $GF(2^k)$

- Addition of two points with $x_1 \neq x_2$

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

- Doubling of a point with $x_1 \neq 0$

$$(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$$

$$\lambda = x_1 + \frac{y_1}{x_1}$$

$$x_3 = \lambda^2 + \lambda + a$$

$$y_3 = x_1^2 + (\lambda + 1)x_3$$

28

Elliptic Curve Cryptosystems

Based on the difficulty of computing the integer e given two points P and eP on the curve

Example: Elliptic Curve Diffie-Hellman

Alice and Bob agree on, the elliptic curve E , the underlying field $GF(2^k)$ or $GF(p)$, and the generating point P with order n

- Alice sends $Q = aP$ to Bob
- Bob sends $R = bP$ to Alice
- Alice computes $S = a(R) = abP$
- Bob computes $S = b(Q) = abP$

Adversary sees P , $Q = aP$, and $R = bP$

Computing $S = abP$ seems to require elliptic logarithms

29

Elliptic Curve Arithmetic

Computation of eP is similar to computation of M^e , which can be performed using exponentiation algorithms

In order to compute e multiple of P we perform elliptic curve additions

An elliptic curve addition is performed by using a few **finite field** operations

Implementation of elliptic curve addition operation requires implementation of three basic finite field operations: **addition, multiplication, and inversion**

For example, addition of two distinct points requires 2 multiplications, 1 inversion, and 9 additions

30

Addition-Subtraction Chains

An addition-subtraction chain is a sequence of integers

$$a_0 \ a_1 \ a_2 \ \cdots \ a_r$$

starting from ± 1 and ending with e such that any a_k is the sum or the difference of two earlier integers a_i and a_j in the chain:

$$a_k = a_i \pm a_j \quad \text{for } 0 < i, j < k$$

Example: $e = 55$

$$\pm 1 \ 2 \ 4 \ 8 \ 7 \ 14 \ 28 \ 56 \ 55$$

An addition-subtraction chain is an algorithm for computing M^e or eP given the integer e

However, it requires negative powers of M or negative multiples of P

31

Recoding Technique

We obtain a sparse signed-digit representation of the exponent with digits $\{0, 1, -1\}$, e.g.,

$$30 = (011110) = 2^4 + 2^3 + 2^2 + 2^1$$

$$30 = (1000\bar{1}0) = 2^5 - 2^1$$

In RSA: we need $M^{-1} \pmod{n}$ as input

Addition-subtraction chains are suitable for elliptic curves since computing $-P$ is trivial

For elliptic curves over $GF(p)$:
if $P = (x, y)$, then $-P = (x, p - y)$

Non-supersingular elliptic curves over $GF(2^k)$:
if $P = (x, y)$, then $-P = (x, x + y)$

32

Exponentiation using Recoding

function $Q := eP$

1. Obtain a signed-digit recoding f of e
2. **if** $f_k = 1$ **then** $Q := P$ **else** $Q := O$
3. **for** $i = k - 1$ **downto** 0
 - 3a. $Q := Q + Q$
 - 3b. **if** $f_i = 1$ **then** $Q := Q + P$
 - 3c. **if** $f_i = \bar{1}$ **then** $Q := Q + (-P)$
4. **return** Q

If the digits of the canonically recoded exponent are scanned w at a time, we obtain the canonical recoding m -ary method

	standard	recoding
binary	1.500 k	1.333 k
quaternary	1.375 k	1.333 k
octal	1.292 k	1.278 k
hex	1.234 k	1.229 k

33

Projective Coordinates

Inversion is a relatively expensive operation

Projective coordinates eliminate the need for performing inversion

In projective coordinates, a point on E has 3 coordinate values

$$(x_1 : y_1 : z_1)$$

while the affine coordinates requires only two values: (x_1, y_1)

Given the distinct points P and Q expressed in projective coordinates

$$P = (x_1 : y_1 : z_1)$$

$$Q = (x_2 : y_2 : z_2)$$

We compute the projective coordinates of the elliptic sum

$$P + Q = (x_3 : y_3 : z_3)$$

34

Projective Coordinates

The projective addition formulae

$$A = x_2 z_1 + x_1$$

$$B = y_2 z_1 + y_1$$

$$C = A + B$$

$$D = A^2(A + az_1) + z_1 BC$$

$$x_3 = AD$$

$$y_3 = CD + A^2(Bx_1 + Ay_1)$$

$$z_3 = A^3 z_1$$

This computation requires 13 field multiplications, and no inversion

35

Projective Coordinates

Similarly, the addition formulae for computing $2P$ is given as

$$A = x_1 z_1$$

$$B = bz_1^4 + x_1^4$$

$$x_3 = AB$$

$$y_3 = x_1^4 A + B(x_1^2 + y_1 z_1 + A)$$

$$z_3 = A^3$$

This computation requires 7 field multiplications, and no inversion

Thus, we have eliminated the inversions at the expense of

- storing 3 $GF(2^k)$ values to represent P
- performing a few more multiplications

36

A Novel Projection Method

Let r be a fixed element of the underlying field $GF(2^k)$ or $GF(p)$

Instead of working with $P = (x, y)$ in order to compute eP , we will work with the image P' under the mapping

$$\begin{aligned} P' &= (x', y') \\ &= (x \cdot r, y \cdot r) \end{aligned}$$

The Menezes projection can be combined with this projection method as well

$$\begin{aligned} P' &= (x' : y' : z') \\ &= (x \cdot r : y \cdot r : z \cdot r) \end{aligned}$$

37

Field Operations

The field operations are modified

Field Addition:

$$\begin{aligned} x' \oplus y' &= x' + y' \\ &= (x \cdot r) + (y \cdot r) \\ &= (x + y) \cdot r \end{aligned}$$

Field Multiplication:

$$\begin{aligned} x' \otimes y' &= x' \cdot y' \cdot r^{-1} \\ &= (x \cdot r) \cdot (y \cdot r) \cdot r^{-1} \\ &= (xy) \cdot r \end{aligned}$$

Field Inversion:

$$\begin{aligned} \text{INV}(x') &= (x')^{-1} \cdot r^2 \\ &= (x \cdot r)^{-1} \cdot r^2 \\ &= (x^{-1}) \cdot r \end{aligned}$$

38

Elliptic Curve Operations

Transform (x, y) to obtain (x', y') for every initial point

Addition of two distinct points

$$\begin{aligned} z' &= \text{INV}(x'_1 + x'_2) \\ L' &= (y'_1 + y'_2) \otimes (z') \\ x'_3 &= (L') \otimes (L') + L' + x'_1 + x'_2 + a' \\ y'_3 &= (L') \otimes (x'_1 + x'_3) + x'_3 + y'_1 \end{aligned}$$

Doubling of a point

$$\begin{aligned} z' &= \text{INV}(x'_1) \\ L' &= x'_1 + (y'_1) \otimes (z') \\ x'_3 &= (L') \otimes (L') + L' + a' \\ y'_3 &= (x'_1) \otimes (x'_1) + (L') \otimes (x'_3) + x'_3 \end{aligned}$$

39

Efficient Implementation

- Field operations: same or easier
 - Addition: Same
 - Multiplication: Easier
 - Inversion: Easier
- Table lookup techniques to speedup word-level multiplications
- Menezes projection to avoid field inversions
- Canonical recoding technique to accelerate exponentiation
- Machine level implementation of basic sub-routines (or primitive operations)

40