Elliptic-curve cryptography
Scalar multiplication, and timing attacks

Tanja Lange

Eindhoven University of Technology

2MMC10 – Cryptology

Double-and-add method

How to compute aP?

```
a = 44444 # our super secret scalar. No, not that one.
l = a.nbits()
A = a.bits()
R = P
for i in range(l-2,-1,-1):
  R = 2 R
  if A[i] == 1:
    R = R + P
print(R)
```

Double-and-add method

How to compute aP?

```
a = 44444 # our super secret scalar. No, not that one.
l = a.nbits()
A = a.bits()
R = P
for i in range(l-2,-1,-1):
  R = 2 R
  if A[i] == 1:
    R = R + P
print(R)
```

This is basically Horner's rule. E.g. $a = 11 = 2^3 + 2 + 1 = (1011)_2$.

$i = 2$: bit is 0, $R = 2P$.

Double-and-add method

How to compute aP?

```
a = 44444 # our super secret scalar. No, not that one.
l = a.nbits()
A = a.bits()
R = P
for i in range(l-2,-1,-1):
  R = 2 R
  if A[i] == 1:
    R = R + P
print(R)
```

This is basically Horner's rule. E.g. $a = 11 = 2^3 + 2 + 1 = (1011)_2$.

$i = 2$: bit is 0, $R = 2P$.
$i = 1$: bit is 1; $R = 4P$; $R = 4P + P = 5P$.

Double-and-add method

How to compute aP?

```
a = 44444 # our super secret scalar. No, not that one.
l = a.nbits()
A = a.bits()
R = P
for i in range(l-2,-1,-1):
  R = 2 R
  if A[i] == 1:
    R = R + P
print(R)
```

This is basically Horner's rule. E.g. $a = 11 = 2^3 + 2 + 1 = (1011)_2$.

$i = 2$: bit is 0, $R = 2P$.
$i = 1$: bit is 1; $R = 4P$; $R = 4P + P = 5P$.
$i = 0$: bit is 1; $R = 10P$; $R = 10P + P = 11P$.

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC`, . . .

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try AAA, BBB, CCC, . . . , CCC takes slightly longer to fail.
Try CAA,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC`, . . . , `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC`, . . . , `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, . . .

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC`, . . . , `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, . . . , `CRR` takes slightly longer to fail.
Try `CRA`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try AAA, BBB, CCC, . . . , CCC takes slightly longer to fail.
Try CAA, CBB, CCC, . . . , CRR takes slightly longer to fail.
Try CRA, CRB,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC`, ..., `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, ..., `CRR` takes slightly longer to fail.
Try `CRA`, `CRB`, `CRC`, ...

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC`, . . . , `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, . . . , `CRR` takes slightly longer to fail.
Try `CRA`, `CRB`, `CRC`, . . . , `CRY` takes slightly longer to fail.

⋮

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC`, ..., `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, ..., `CRR` takes slightly longer to fail.
Try `CRA`, `CRB`, `CRC`, ..., `CRY` takes slightly longer to fail.

⋮

Password is `CRYPTOLOGY`.

1974: Exploit developed by Alan Bell for TENEX operating system.

Reminder: double-and-add method

Compute aP given a and P.

```
a = 44444 # our super secret scalar. No, not that one.
l = a.nbits()
A = a.bits()
R = P
for i in range(l-2,-1,-1):
  R = 2 R
  if A[i] == 1:
    R = R + P
print(R)
```

Reminder: double-and-add method

Compute aP given a and P.

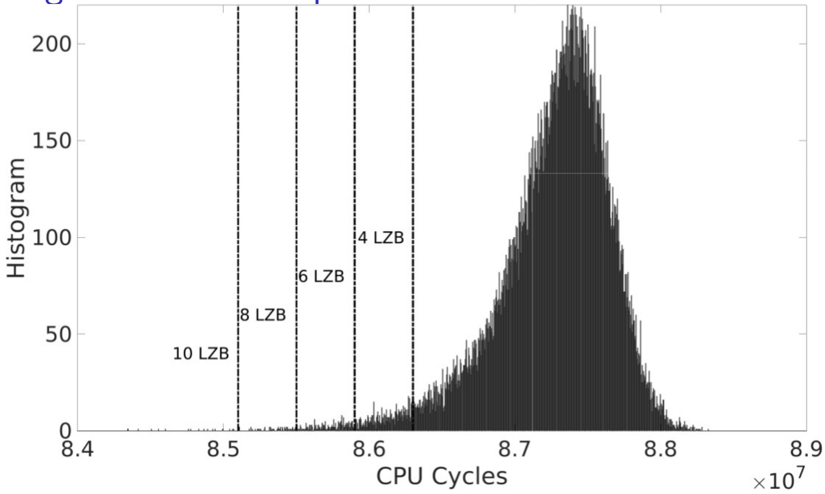
```
a = 44444 # our super secret scalar. No, not that one.
l = a.nbits()
A = a.bits()
R = P
for i in range(l-2,-1,-1):   # loop length depends on a
  R = 2 R
  if A[i] == 1:
    R = R + P
print(R)
```

Reminder: double-and-add method

Compute aP given a and P.

```
a = 44444 # our super secret scalar. No, not that one.
l = a.nbits()
A = a.bits()
R = P
for i in range(l-2,-1,-1):   # loop length depends on a
  R = 2 R
  if A[i] == 1:    # branch depends on a
    R = R + P
print(R)
```

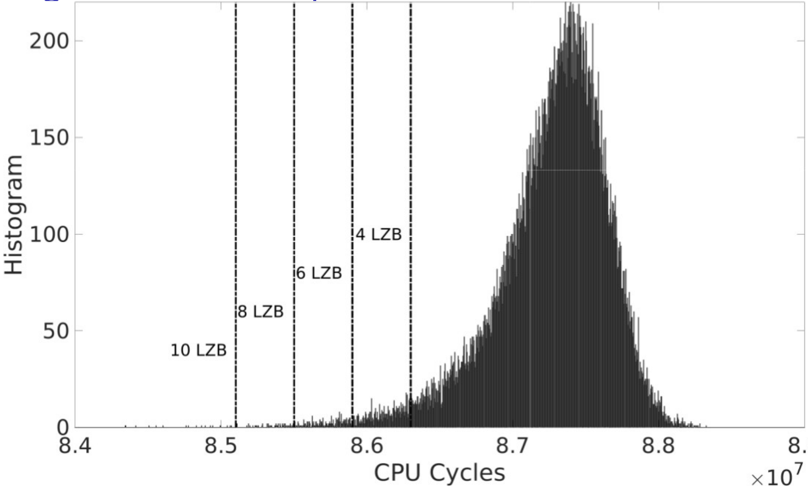
Timings of scalar multiplication on NIST P-256



(Picture from TPM-Fail)

NIST P-256 is an elliptic curve standardized by NIST.
It is a Weierstrass curve modulo a 256-bit prime.

Timings of scalar multiplication on NIST P-256



(Picture from TPM-Fail)

NIST P-256 is an elliptic curve standardized by NIST.
It is a Weierstrass curve modulo a 256-bit prime.
Timing depends strongly on the length of the scalar, also on Hamming weight.

Fixed window method

- Faster methods reduce the number of additions by using windows: $14019 =$

Fixed window method

- Faster methods reduce the number of additions by using windows:
  14019 =11 0110 1100 0011

Fixed window method

- Faster methods reduce the number of additions by using windows:
  $14019 = \underbrace{11}_{3}\ \underbrace{0110}_{1\ 2}\ \underbrace{1100}_{3\ 0}\ \underbrace{0011}_{0\ 3}$

Fixed window method

- Faster methods reduce the number of additions by using windows:
  $14019 = \underbrace{11}_{3}\,\underbrace{0110}_{1\ 2}\,\underbrace{1100}_{3\ 0}\,\underbrace{0011}_{0\ 3}$
  Precompute P, $2P$, and $3P$. Left window is innermost coefficient.

  $$14019P = 4\left(4\left(4\left(4\left(4\left(4\left(3P\right)+P\right)+2P\right)+3P\right)\right)\right)+3P.$$

  Same number of doublings, 4 instead of 7 additions.

Fixed window method

- Faster methods reduce the number of additions by using windows:
  $14019 = \underbrace{11}_{3}\ \underbrace{0110}_{1\ 2}\ \underbrace{1100}_{3\ 0}\ \underbrace{0011}_{0\ 3}$
  Precompute P, $2P$, and $3P$. Left window is innermost coefficient.

  $$14019P = 4\left(4\left(4\left(4\left(4\left(4\left(3P\right)+P\right)+2P\right)+3P\right)\right)\right)+3P.$$

  Same number of doublings, 4 instead of 7 additions.
- General case: width-w windows.
  Start from least-significant bit (coefficient of 2^0)
  turn w bits into coefficient in $[2^w - 1, 0]$,
  pad with 0 bits if length is not a multiple of w.

Fixed window method

- Faster methods reduce the number of additions by using windows:
  $14019 = \underbrace{11}_{3}\ \underbrace{0110}_{1\ 2}\ \underbrace{1100}_{3\ 0}\ \underbrace{0011}_{0\ 3}$
  Precompute P, $2P$, and $3P$. Left window is innermost coefficient.

  $$14019P = 4\,(4\,(4\,(4\,(4\,(4\,(3P) + P) + 2P) + 3P))) + 3P.$$

  Same number of doublings, 4 instead of 7 additions.
- General case: width-w windows.
  Start from least-significant bit (coefficient of 2^0)
  turn w bits into coefficient in $[2^w - 1, 0]$,
  pad with 0 bits if length is not a multiple of w.

  E.g. $w = 4$, so coefficients in $[15, 0]$.
  $14019 = \underbrace{0011}_{3}\ \underbrace{0110}_{6}\ \underbrace{1100}_{12}\ \underbrace{0011}_{3}$

Fixed window method

- Faster methods reduce the number of additions by using windows:
  $14019 = \underbrace{11}_{3}\ \underbrace{0110}_{1\ 2}\ \underbrace{1100}_{3\ 0}\ \underbrace{0011}_{0\ 3}$
  Precompute P, $2P$, and $3P$. Left window is innermost coefficient.

  $$14019P = 4\left(4\left(4\left(4\left(4\left(4\left(3P\right) + P\right) + 2P\right) + 3P\right)\right)\right) + 3P.$$

  Same number of doublings, 4 instead of 7 additions.

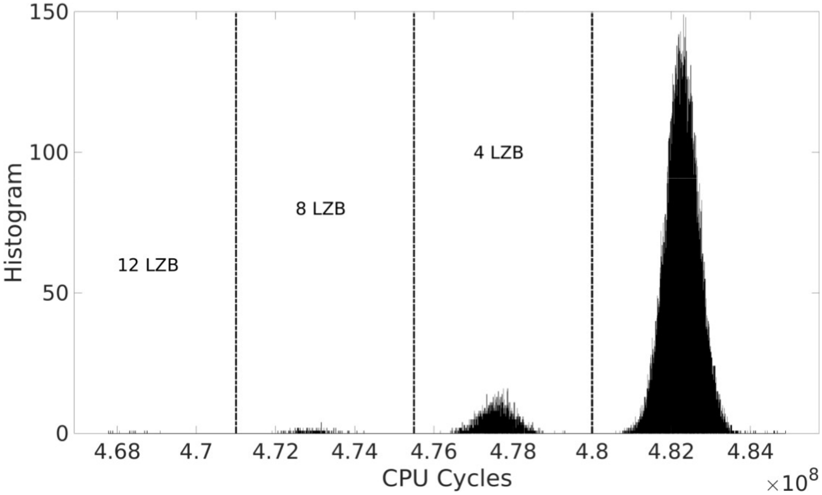
- General case: width-w windows.
  Start from least-significant bit (coefficient of 2^0)
  turn w bits into coefficient in $[2^w - 1, 0]$,
  pad with 0 bits if length is not a multiple of w.

  E.g. $w = 4$, so coefficients in $[15, 0]$.
  $14019 = \underbrace{0011}_{3}\ \underbrace{0110}_{6}\ \underbrace{1100}_{12}\ \underbrace{0011}_{3}$

  $$14019P = 16\left(16\left(16\left(3P\right) + 6P\right) + 12P\right) + 3P.$$

  Same number of doublings, 3 additions.

Timings of scalar multiplication on NIST P-256



Larger windows reduce the variability through branching but accentuate the length.

(Picture from TPM-Fail)

Double-and-always-add

```
a = 44444   # our super secret scalar. No, not that one.
l = max    # some maximum bit length, matching order(P)
A = a.digits(2,padto = l) # fill with 0 to lenght l
R = 0     # so initial doublings don't matter, 0=0P
for i in range(l-1,-1,-1): # fixed-length loop
  R = 2R
  Q = R + P
  R = (1 - A[i]) * R + A[i] * Q # selection by arithmetic
print(R)
```

This costs 1 addition per bit, so as slow as worst case,
but leads to uniform trace – if the other operations are uniform.

Double-and-always-add

```
a = 44444   # our super secret scalar. No, not that one.
l = max    # some maximum bit length, matching order(P)
A = a.digits(2,padto = l) # fill with 0 to lenght l
R = 0     # so initial doublings don't matter, 0=0P
for i in range(l-1,-1,-1): # fixed-length loop
  R = 2R
  Q = R + P
  R = (1 - A[i]) * R + A[i] * Q # selection by arithmetic
print(R)
```

This costs 1 addition per bit, so as slow as worst case,
but leads to uniform trace – if the other operations are uniform.

- Formulas for addition on Weierstrass curves have exceptions for adding ∞, so initialization at ∞ does not work.
- Edwards curves have a complete addition law, **easy** to double or add the neutral element $(0, 1)$.

Montgomery ladder

```
def cswap(bit, R, S):  # constant time conditional swap
        dummy = bit * (R - S)   # 0 or R - S
        R = R - dummy     # R or R - (R - S) = S
        S = S + dummy     # S or S + (R - S) = R
        return (R, S)
```

```
a = 44444   # our super secret scalar. No, not that one.
l = max    # some maximum bit length, matching order(P)
A = a.digits(2,padto = l) # fill with 0 to lenght l
P0 = 0     # so initial doublings don't matter, 0=0P
P1 = P    # difference P1 - P0 = P
for i in range(l-1,-1,-1): # fixed-length loop
  (P0, P1) = cswap(A[i], P0, P1)  # see above
  P1 = P0 + P1  # addition with fixed difference
  P0 = 2P0  # double point for which bit is set
  (P0, P1) = cswap(A[i], P0, P1) # swap back, can merge
print(P0)
```

This uses one doubling and one addition per bit. No dummy additions.

Loop in Montgomery ladder

```
P0 = 0     # so initial doublings don't matter, 0=0P
P1 = P    # difference P1 - P0 = P
for i in range(l-1,-1,-1): # fixed-length loop
  (P0, P1) = cswap(A[i], P0, P1)  # see above
  P1 = P0 + P1  # addition with fixed difference
  P0 = 2P0  # double point for which bit is set
  (P0, P1) = cswap(A[i], P0, P1) # swap back, can merge
print(P0)
```

Loop in Montgomery ladder

```
P0 = 0     # so initial doublings don't matter, 0=0P
P1 = P    # difference P1 - P0 = P
for i in range(l-1,-1,-1): # fixed-length loop
  (P0, P1) = cswap(A[i], P0, P1)  # see above
  P1 = P0 + P1  # addition with fixed difference
  P0 = 2P0  # double point for which bit is set
  (P0, P1) = cswap(A[i], P0, P1) # swap back, can merge
print(P0)
```

if `A[i]=0`:
`cswap(A[i], P0, P1)` leaves fixed,
so the new values are
`P0 = 2P0, P1 = P0 + P1`
(no effect of swapping back).

if `A[i]=1`:
`cswap(A[i], P0, P1)` swaps,
so the new values are
`P1 = 2P1, P0 = P0 + P1`
(after swapping back).

Loop in Montgomery ladder

```
P0 = 0     # so initial doublings don't matter, 0=0P
P1 = P    # difference P1 - P0 = P
for i in range(l-1,-1,-1): # fixed-length loop
  (P0, P1) = cswap(A[i], P0, P1)  # see above
  P1 = P0 + P1  # addition with fixed difference
  P0 = 2P0  # double point for which bit is set
  (P0, P1) = cswap(A[i], P0, P1) # swap back, can merge
print(P0)
```

if `A[i]=0`:
`cswap(A[i], P0, P1)` leaves fixed,
so the new values are
`P0 = 2P0, P1 = P0 + P1`
(no effect of swapping back).

if `A[i]=1`:
`cswap(A[i], P0, P1)` swaps,
so the new values are
`P1 = 2P1, P0 = P0 + P1`
(after swapping back).

Either way, `P1 - P0 = P` after each step.

Addition is of points with know difference called differential addition.
This uses one doubling and one differential addition per bit.

Montgomery differential addition

Let $nP = (U_n : V_n : Z_n), mP = (U_m : V_m : Z_m)$ with known difference $(m-n)P = (U_{m-n} : V_{m-n} : Z_{m-n})$ on

$$M_{A,B} : Bv^2 = u^3 + Au^2 + u.$$

We will only use U and Z; cheaper by skipping V.

Montgomery differential addition

Let $nP = (U_n : V_n : Z_n), mP = (U_m : V_m : Z_m)$ with known difference $(m-n)P = (U_{m-n} : V_{m-n} : Z_{m-n})$ on

$$M_{A,B} : Bv^2 = u^3 + Au^2 + u.$$

We will only use U and Z; cheaper by skipping V.

Addition: $n \neq m$

$$\begin{aligned} U_{m+n} &= Z_{m-n}\big((U_m - Z_m)(U_n + Z_n) + (U_m + Z_m)(U_n - Z_n)\big)^2, \\ Z_{m+n} &= U_{m-n}\big((U_m - Z_m)(U_n + Z_n) - (U_m + Z_m)(U_n - Z_n)\big)^2 \end{aligned}$$

Doubling: $n = m$

$$\begin{aligned} 4U_nZ_n &= (U_n + Z_n)^2 - (U_n - Z_n)^2, \\ U_{2n} &= (U_n + Z_n)^2(U_n - Z_n)^2, \\ Z_{2n} &= 4U_nZ_n\big((U_n - Z_n)^2 + \big((A+2)/4\big)(4U_nZ_n)\big). \end{aligned}$$

Differential addition takes 4M and 2S. Doubling takes 3M and 2S.

Montgomery differential addition

Let $nP = (U_n : V_n : Z_n), mP = (U_m : V_m : Z_m)$ with known difference $(m-n)P = (U_{m-n} : V_{m-n} : Z_{m-n})$ on

$$M_{A,B} : Bv^2 = u^3 + Au^2 + u.$$

We will only use U and Z; cheaper by skipping V.

Addition: $n \neq m$

$$\begin{aligned} U_{m+n} &= Z_{m-n}\big((U_m - Z_m)(U_n + Z_n) + (U_m + Z_m)(U_n - Z_n)\big)^2, \\ Z_{m+n} &= U_{m-n}\big((U_m - Z_m)(U_n + Z_n) - (U_m + Z_m)(U_n - Z_n)\big)^2 \end{aligned}$$

Doubling: $n = m$

$$\begin{aligned} 4U_nZ_n &= (U_n + Z_n)^2 - (U_n - Z_n)^2, \\ U_{2n} &= (U_n + Z_n)^2(U_n - Z_n)^2, \\ Z_{2n} &= 4U_nZ_n\big((U_n - Z_n)^2 + \big((A+2)/4\big)(4U_nZ_n)\big). \end{aligned}$$

Differential addition takes 4M and 2S. Doubling takes 3M and 2S.
In ladder, $m - n = 1$, choose $Z_{m-n} = 1$ and $(A+2)/4$ small.
Then cost per bit: 5M and 4S. Also like U_{m-n} small.

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1.$

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A + 2)/4 = 121666$ is smallest with all properties from http://safecurves.cr.yp.to/.

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1.$

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A + 2)/4 = 121666$ is smallest with all properties from
`http://safecurves.cr.yp.to/`.

This curve is birationally equivalent to Edwards curve

$$x^2 + y^2 = 1 + dx^2y^2 \text{ for } d = 121665/121666.$$

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1.$

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A + 2)/4 = 121666$ is smallest with all properties from
http://safecurves.cr.yp.to/.

This curve is birationally equivalent to Edwards curve

$$x^2 + y^2 = 1 + dx^2y^2 \text{ for } d = 121665/121666.$$

Note that the map given in part VI maps to $a'x'^2 + y'^2 = 1 + d'x'^2y'^2$ with $a' = 486664, d' = 486660.$

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1.$

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A+2)/4 = 121666$ is smallest with all properties from
`http://safecurves.cr.yp.to/`.

This curve is birationally equivalent to Edwards curve

$$x^2 + y^2 = 1 + dx^2y^2 \text{ for } d = 121665/121666.$$

Note that the map given in part VI maps to $a'x'^2 + y'^2 = 1 + d'x'^2y'^2$
with $a' = 486664, d' = 486660.$
Note $a' = b^2$ in F_p and change $x = bx'$, $y = y'$.

Example: Curve25519 (Bernstein 2006)

Let $p = 2^{255} - 19, A = 486662, B = 1.$

$$v^2 = u^3 + 486662u^2 + u$$

Is standardized for DH computations for the Internet in RFC 7748

$(A + 2)/4 = 121666$ is smallest with all properties from
http://safecurves.cr.yp.to/.

This curve is birationally equivalent to Edwards curve

$$x^2 + y^2 = 1 + dx^2y^2 \text{ for } d = 121665/121666.$$

Note that the map given in part VI maps to $a'x'^2 + y'^2 = 1 + d'x'^2y'^2$
with $a' = 486664, d' = 486660.$
Note $a' = b^2$ in F_p and change $x = bx'$, $y = y'$.
This maps to $x^2 + y^2 = 1 + dx^2y^2$ with $d = d'/a' = 121665/121666.$