Elliptic-curve cryptography VIII

Constant-time scalar multiplication

Tanja Lange

Eindhoven University of Technology

2MMC10 – Cryptology

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.
Differential addition is faster than general addition on some curves incl. Montgomery curves (see part VIII).