Projekt: Kryptografie mit elliptischen Kurven

Leon Groß 108018221971

Pseudocodes zu tatsächlicher Implementierung

June 18, 2019

Algorithm 1: Modular Addition

input : integer a, integer b, natural number module
output: modular added values [a + b mod module]

return *((a mod module) + (b mod moduel)) mod module*

Algorithm 2: Modular Subtraction

input : integer a, intger b, natural number module
output: modular subtracted values [a - b mod module]

return *((a mod module) - (b mod moudle)) mod module*

Algorithm 3: Modular Multiplication

input : integer a, inetger b, natural number module
output: modular muliplied values [(a * b) mod module]

$resuslt \leftarrow 0$
for $i \leftarrow 0$ **to** $b - 1$ **do**
    $result = \mod\_add(result, b, module)$
    $result = resultmodmodule$
return $result$

Algorithm 4: Extended Euklidian Algrorithm (recursive)

input : integer a, natural number b
output: Touple d,s,t ,result of the internally calculated table

if b = *0* **then**
  return $Touple(a, 1, 0)$; `// return value without further compuattion`

$d1, s1, t1 \leftarrow$ extended_euclidian_algorithm_rec(b, a mod b)
$d \leftarrow d1$
$s \leftarrow t1$
$t \leftarrow s1 - (a/b) * t1$
return d, s, t

Algorithm 5: Extended Euklidian Algorithm Wrapper for recursive function

input : integer a, natural number b
output: modular inverse of a in ${GF_2}^b$

$result \leftarrow$ extended_euclidian_algorithm_rec(a, b)
return *s from result Touple mod b*

Algorithm 6: Square-And-Multiply

input : inetger base, int exp, natural number module
output: $base^{exp}$ mod module

exp_bin $\leftarrow$ exponent in binary form ; `// binary exponent without righest 1`
$result \leftarrow base$; `// initial value`
foreach *integer of exp_bin* **do**
  $result \leftarrow result^2$ mod module
  if *elem* = *1* **then**
    $result \leftarrow result * base$ mod module

return *result*

Algorithm 7: Fermats Little Theorem

input : integer a, natural prime number p
output: modular inverse of a in ${GF_2}^p$

return *sqaure_and_multiply*$(a, p - 2, p)$
; `// This function makes us of the frequently introduced function`
; `//` *sqaure_and_multiply* `- fast exponentation is needed`

Algorithm 8: Double Point

input : Point P, elliptic curve E, inversion mode
output: Point Q = P + P

$s \leftarrow inversion(2*\mathrm{y}_P,\ mod_E)_{mode}$ * ($3*(x_P)^2 + a_E$ mod mod_E
$x_3 \leftarrow s^2 - x_P - x_P$ mod mod_E
$y_3 \leftarrow s * (x_P - x_3) - y_P$ mod mod_E
return *Point(*x_3, y_3*)*

Algorithm 9: Add points

input : Point P, Point Q, elliptic curve E, inversion mode
output: Point R = P + Q

$s \leftarrow ((y_Q - y_P)$ mod $mod_E * inversion(x_Q - x_P, mod_E)_{mode})$ mod mod_E
$x_3 \leftarrow s^2 - x_P - x_Q$ mod mod_E
$y_3 \leftarrow s * (x_P - x_3) - y_P$ mod mod_E
return *Point(*x_3, y_3*)*

Algorithm 10: Multiply Points

input : natural skalar factor k, Point P, elliptic curve E, inversion mode
output: T = k * P

bin_exp ←binary representation of k without first 1
Point T ← P
foreach *integer of bin_exp* **do**
    $T \leftarrow double\_point(T, E, mode_{inversion})$
    if *elem = 1* **then**
        $T \leftarrow$ add_points(T, P, E, mode$_{inversion}$)
return T

Algorithm 11: NAF Point Multiplication

input : natural skalar factor k, Point P, elliptic curve E, inversion mode
output: Point Q = k * P

$Q \leftarrow PointP(0,0)$
$naf\_exp \leftarrow$ NAF(k)
foreach *integer of naf_exp* **do**
  $Q \leftarrow$ multiply_points(2, Q, E, mode$_{inversion}$)
  if *elem = 1* **then**
    $Q \leftarrow$ add_points(Q, P, E, mode$_{inversion}$)
  if *elem = -1* **then**
    $Q \leftarrow$ add_points(Q, P$^{-1}, E, mode_{inversion}$)
return Q

Algorithm 12: Calculate AES Key (res= 0x9a19f8e811c45299cb1e6625562f8505)

input : T$_{AB}$
output: AES key (base 16)

$T\_AB\_bin \leftarrow$ binary representation of T$_{AB}$
$T\_AB\_bin\_1 \leftarrow$ bits b_0 to $b_1$28 of T_AB_bin
$T\_AB\_bin\_2 \leftarrow$ bits $b_1$28 to $b_2$56 of T_AB_bin
return (T_AB_bin_1 $\oplus$ T_AB_bin_2)$_{16}$