Project5 SM2算法的实现和优化

密码22.1 张靖陶 202200460011

1. SM2算法结构

(1) 椭圆曲线参数

使用标准化的椭圆曲线参数：素数域p、曲线系数a、b

基点G(Gx, Gy)作为生成元

素数阶n（基点G的阶）

(2) 密钥对生成

随机生成私钥d ∈ [1, n-1]

计算公钥P = [d]G（标量乘法）

公钥为椭圆曲线上的点(x, y)

(3) 加密流程

输入：接收方公钥P，明文消息M

生成随机数k ∈ [1, n-1]

计算：

C1 = [k]G→密文第一部分

S = [k]P→共享密钥点

(x2, y2) = S的坐标

密钥派生：

t = KDF(x2||y2, len(M)) → 派生对称密钥

加密：

C2 = M ⊕ t → 密文第二部分

完整性验证：

C3 = SM3(x2||M||y2) → 哈希值

输出：密文C = C1||C3||C2

(4) 解密流程

输入：私钥d，密文C

C1 = 前65字节 → 椭圆曲线点

C3 = 接下来32字节 → 哈希值

C2 = 剩余部分 → 加密数据

计算：

S = [d]C1 → 共享密钥点

(x2, y2) = S的坐标

密钥派生：

t = KDF(x2||y2, len(C2))

解密：

M' = C2 ⊕ t

完整性验证：

计算u = SM3(x2||M'||y2)

验证u == C3

输出：明文M'

1. 代码实现

|  |
| --- |
| import os |
| import time |
| import struct |
| from typing import Tuple, Optional |
|  |
| # 椭圆曲线参数 |
| p = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF |
| a = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC |
| b = 0x28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93 |
| n = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123 |
| Gx = 0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7 |
| Gy = 0xBC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0 |
|  |
| # 辅助函数 |
| def int\_to\_bytes(x: int, length: int) -> bytes: |
| return x.to\_bytes(length, 'big') |
|  |
| def bytes\_to\_int(b: bytes) -> int: |
| return int.from\_bytes(b, 'big') |
|  |
| def mod\_inv(x: int, m: int = p) -> int: |
| return pow(x, m - 2, m) |
|  |
| # 椭圆曲线运算 |
| Point = Optional[Tuple[int, int]] |
| O: Point = None |
|  |
| def is\_on\_curve(P: Point) -> bool: |
| if P is None: return True |
| x, y = P |
| return (y \* y - (x \* x \* x + a \* x + b)) % p == 0 |
|  |
| def point\_add(P: Point, Q: Point) -> Point: |
| if P is None: return Q |
| if Q is None: return P |
| x1, y1 = P |
| x2, y2 = Q |
| if x1 == x2 and (y1 + y2) % p == 0: |
| return None |
| if P != Q: |
| lam = ((y2 - y1) \* mod\_inv((x2 - x1) % p, p)) % p |
| else: |
| lam = ((3 \* x1 \* x1 + a) \* mod\_inv((2 \* y1) % p, p)) % p |
| x3 = (lam \* lam - x1 - x2) % p |
| y3 = (lam \* (x1 - x3) - y1) % p |
| return (x3, y3) |
|  |
| def scalar\_mult(k: int, P: Point) -> Point: |
| if k % n == 0 or P is None: |
| return None |
| if k < 0: |
| return scalar\_mult(-k, (P[0], (-P[1]) % p)) |
| R = None |
| Q = P |
| while k: |
| if k & 1: |
| R = point\_add(R, Q) |
| Q = point\_add(Q, Q) |
| k >>= 1 |
| return R |
|  |
| # SM3 哈希 |
| def \_rotl(x, n): |
| n = n % 32 |
| return ((x << n) & 0xFFFFFFFF) | (x >> (32 - n)) |
|  |
| def \_P0(x: int) -> int: |
| return x ^ \_rotl(x, 9) ^ \_rotl(x, 17) |
|  |
| def \_P1(x: int) -> int: |
| return x ^ \_rotl(x, 15) ^ \_rotl(x, 23) |
|  |
| T\_j = [0x79cc4519] \* 16 + [0x7a879d8a] \* 48 |
|  |
| def sm3\_compress(V: int, B: bytes) -> int: |
| W = [bytes\_to\_int(B[4 \* i:4 \* i + 4]) for i in range(16)] |
| for j in range(16, 68): |
| x = W[j - 16] ^ W[j - 9] ^ \_rotl(W[j - 3], 15) |
| W.append(\_P1(x) ^ \_rotl(W[j - 13], 7) ^ W[j - 6]) |
| W1 = [W[j] ^ W[j + 4] for j in range(64)] |
|  |
| Vs = [(V >> (32 \* (7 - i))) & 0xFFFFFFFF for i in range(8)] |
| A, B1, C, D, E, F, G, H = Vs |
| for j in range(64): |
| if j <= 15: |
| FF = A ^ B1 ^ C |
| GG = E ^ F ^ G |
| else: |
| FF = (A & B1) | (A & C) | (B1 & C) |
| GG = (E & F) | ((~E) & G & 0xFFFFFFFF) |
| SS1 = \_rotl(((\_rotl(A, 12) + E + \_rotl(T\_j[j], j)) & 0xFFFFFFFF), 7) |
| SS2 = SS1 ^ \_rotl(A, 12) |
| TT1 = (FF + D + SS2 + W1[j]) & 0xFFFFFFFF |
| TT2 = (GG + H + SS1 + W[j]) & 0xFFFFFFFF |
| D, C, B1, A = C, \_rotl(B1, 9), A, TT1 |
| H, G, F, E = G, \_rotl(F, 19), E, \_P0(TT2) |
|  |
| Vv = [(Vs[0] ^ A) & 0xFFFFFFFF, (Vs[1] ^ B1) & 0xFFFFFFFF, |
| (Vs[2] ^ C) & 0xFFFFFFFF, (Vs[3] ^ D) & 0xFFFFFFFF, |
| (Vs[4] ^ E) & 0xFFFFFFFF, (Vs[5] ^ F) & 0xFFFFFFFF, |
| (Vs[6] ^ G) & 0xFFFFFFFF, (Vs[7] ^ H) & 0xFFFFFFFF] |
| res = 0 |
| for x in Vv: |
| res = (res << 32) | (x & 0xFFFFFFFF) |
| return res |
|  |
| def sm3\_hash(msg: bytes) -> bytes: |
| IV = bytes.fromhex('7380166f4914b2b9172442d7da8a0600a96f30bc163138aae38dee4db0fb0e4e') |
| msg\_len = len(msg) |
| bit\_len = msg\_len \* 8 |
| msg\_padded = msg + b'\x80' |
| k = (56 - (len(msg\_padded) % 64)) % 64 |
| msg\_padded += b'\x00' \* k |
| msg\_padded += struct.pack('>Q', bit\_len) |
| V = int.from\_bytes(IV, 'big') |
| for i in range(0, len(msg\_padded), 64): |
| block = msg\_padded[i:i + 64] |
| V = sm3\_compress(V, block) |
| return int\_to\_bytes(V, 32) |
|  |
| def kdf(z: bytes, klen: int) -> bytes: |
| ct = 1 |
| out = b'' |
| while len(out) < klen: |
| out += sm3\_hash(z + struct.pack('>I', ct)) |
| ct += 1 |
| return out[:klen] |
|  |
| def generate\_keypair() -> Tuple[int, Tuple[int, int]]: |
| while True: |
| d = bytes\_to\_int(os.urandom(32)) % n |
| if 1 <= d < n: |
| break |
| P = scalar\_mult(d, (Gx, Gy)) |
| return d, P |
|  |
| def point\_to\_bytes(P: Point) -> bytes: |
| if P is None: |
| raise ValueError("无穷远点") |
| return b'\x04' + int\_to\_bytes(P[0], 32) + int\_to\_bytes(P[1], 32) |
|  |
| def bytes\_to\_point(b: bytes) -> Point: |
| if b[0] != 4: |
| raise ValueError("只支持未压缩点") |
| x = bytes\_to\_int(b[1:33]) |
| y = bytes\_to\_int(b[33:65]) |
| P = (x, y) |
| if not is\_on\_curve(P): |
| raise ValueError("点不在曲线上") |
| return P |
|  |
| def sm2\_encrypt(pub: Tuple[int, int], msg: bytes) -> bytes: |
| mlen = len(msg) |
| while True: |
| k = bytes\_to\_int(os.urandom(32)) % n |
| if k == 0: |
| continue |
| C1 = scalar\_mult(k, (Gx, Gy)) |
| S = scalar\_mult(k, pub) |
| x2, y2 = int\_to\_bytes(S[0], 32), int\_to\_bytes(S[1], 32) |
| t = kdf(x2 + y2, mlen) |
| if int.from\_bytes(t, 'big') == 0: |
| continue |
| C2 = bytes([m ^ t\_i for m, t\_i in zip(msg, t)]) |
| C3 = sm3\_hash(x2 + msg + y2) |
| return point\_to\_bytes(C1) + C3 + C2 |
|  |
| def sm2\_decrypt(priv: int, C: bytes) -> bytes: |
| C1 = bytes\_to\_point(C[:65]) |
| C3 = C[65:97] |
| C2 = C[97:] |
| S = scalar\_mult(priv, C1) |
| x2, y2 = int\_to\_bytes(S[0], 32), int\_to\_bytes(S[1], 32) |
| t = kdf(x2 + y2, len(C2)) |
| if int.from\_bytes(t, 'big') == 0: |
| raise ValueError("KDF 输出全 0") |
| M = bytes([c ^ t\_i for c, t\_i in zip(C2, t)]) |
| if sm3\_hash(x2 + M + y2) != C3: |
| raise ValueError("C3 校验失败") |
| return M |
|  |
| # 测试 |
| def bench(rounds: int = 20): |
| print("生成密钥对…") |
| d, P = generate\_keypair() |
| message = b"abc" |
| print("原文:", message) |
|  |
| # 单次验证 |
| C = sm2\_encrypt(P, message) |
| M = sm2\_decrypt(d, C) |
| print("解密结果:", M) |
| print("解密正确:", M == message) |
|  |
| # 加密 |
| t0 = time.perf\_counter() |
| for \_ in range(rounds): |
| sm2\_encrypt(P, message) |
| t1 = time.perf\_counter() |
| print(f"平均加密时间: {(t1 - t0) / rounds \* 1000:.3f} ms") |
|  |
| # 解密 |
| C = sm2\_encrypt(P, message) |
| t0 = time.perf\_counter() |
| for \_ in range(rounds): |
| sm2\_decrypt(d, C) |
| t1 = time.perf\_counter() |
| print(f"平均解密时间: {(t1 - t0) / rounds \* 1000:.3f} ms") |
|  |
| if \_\_name\_\_ == "\_\_main\_\_": |
| bench(10) |

1. 优化思路

原代码在每次加密/解密过程中频繁把大整数（点坐标、临时值）转成 bytes，再把 bytes 再转回整数。int->bytes 和 bytes->int 都会做多次内存分配和大整数到字节的O(w)位拆分运算，在Python中代价很高。

把椭圆曲线与标量运算全部保留为整数（内存里以 Python int 表示），只在输入/输出（或必须传给 SM3/KDF）时一次性做 int->bytes。也就是把多次转换合并成“仅一次”或很少次数。

1. 优化后的代码

|  |
| --- |
| from \_\_future\_\_ import annotations |
| import os |
| import time |
| import struct |
| from typing import Tuple, Optional |
|  |
| # 椭圆曲线参数 |
| p = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF |
| a = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC |
| b = 0x28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93 |
| n = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123 |
| Gx = 0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7 |
| Gy = 0xBC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0 |
|  |
| def int\_to\_bytes(x: int, length: int) -> bytes: |
| return x.to\_bytes(length, 'big') |
|  |
| def bytes\_to\_int(b: bytes) -> int: |
| return int.from\_bytes(b, 'big') |
|  |
| def concat\_ints\_to\_bytes(x: int, y: int) -> bytes: |
| return int\_to\_bytes(x, 32) + int\_to\_bytes(y, 32) |
|  |
| # 模逆 |
| def mod\_inv(x: int, m: int = p) -> int: |
| return pow(x, m - 2, m) |
|  |
| # 椭圆曲线点运算 |
| Point = Optional[Tuple[int, int]] |
| O: Point = None |
|  |
| def is\_on\_curve(P: Point) -> bool: |
| if P is None: |
| return True |
| x, y = P |
| return (y \* y - (x \* x \* x + a \* x + b)) % p == 0 |
|  |
| def point\_add(P: Point, Q: Point) -> Point: |
| if P is None: |
| return Q |
| if Q is None: |
| return P |
| x1, y1 = P |
| x2, y2 = Q |
| if x1 == x2 and (y1 + y2) % p == 0: |
| return None |
| if P != Q: |
| lam = ((y2 - y1) \* mod\_inv((x2 - x1) % p, p)) % p |
| else: |
| lam = ((3 \* x1 \* x1 + a) \* mod\_inv((2 \* y1) % p, p)) % p |
| x3 = (lam \* lam - x1 - x2) % p |
| y3 = (lam \* (x1 - x3) - y1) % p |
| return (x3, y3) |
|  |
| def scalar\_mult(k: int, P: Point) -> Point: |
| if k % n == 0 or P is None: |
| return None |
| if k < 0: |
| return scalar\_mult(-k, (P[0], (-P[1]) % p)) |
| R = None |
| Q = P |
| while k: |
| if k & 1: |
| R = point\_add(R, Q) |
| Q = point\_add(Q, Q) |
| k >>= 1 |
| return R |
|  |
| def point\_to\_bytes(P: Point) -> bytes: |
| if P is None: |
| raise ValueError("无穷远点") |
| x, y = P |
| return b'\x04' + int\_to\_bytes(x, 32) + int\_to\_bytes(y, 32) |
|  |
| def bytes\_to\_point(b: bytes) -> Point: |
| if b[0] != 4: |
| raise ValueError("只支持未压缩点格式 0x04") |
| x = bytes\_to\_int(b[1:33]) |
| y = bytes\_to\_int(b[33:65]) |
| P = (x, y) |
| if not is\_on\_curve(P): |
| raise ValueError("点不在曲线上") |
| return P |
|  |
| # SM3 哈希 |
| def \_rotl(x: int, n: int) -> int: |
| n = n & 31 # 等同 n % 32，但更快 |
| return ((x << n) & 0xFFFFFFFF) | (x >> (32 - n)) |
|  |
| def \_P0(x: int) -> int: |
| return x ^ \_rotl(x, 9) ^ \_rotl(x, 17) |
|  |
| def \_P1(x: int) -> int: |
| return x ^ \_rotl(x, 15) ^ \_rotl(x, 23) |
|  |
| T\_j = [0x79cc4519] \* 16 + [0x7a879d8a] \* 48 |
|  |
| def sm3\_compress(V: int, B: bytes) -> int: |
| W = [bytes\_to\_int(B[4\*i:4\*i+4]) for i in range(16)] |
| for j in range(16, 68): |
| x = W[j-16] ^ W[j-9] ^ \_rotl(W[j-3], 15) |
| W.append(\_P1(x) ^ \_rotl(W[j-13], 7) ^ W[j-6]) |
| W1 = [W[j] ^ W[j+4] for j in range(64)] |
|  |
| Vs = [(V >> (32\*(7-i))) & 0xFFFFFFFF for i in range(8)] |
| A, B1, C, D, E, F, G, H = Vs |
| for j in range(64): |
| if j <= 15: |
| FF = A ^ B1 ^ C |
| GG = E ^ F ^ G |
| else: |
| FF = (A & B1) | (A & C) | (B1 & C) |
| GG = (E & F) | ((~E) & G & 0xFFFFFFFF) |
| SS1 = \_rotl(((\_rotl(A, 12) + E + \_rotl(T\_j[j], j)) & 0xFFFFFFFF), 7) |
| SS2 = SS1 ^ \_rotl(A, 12) |
| TT1 = (FF + D + SS2 + W1[j]) & 0xFFFFFFFF |
| TT2 = (GG + H + SS1 + W[j]) & 0xFFFFFFFF |
| D = C |
| C = \_rotl(B1, 9) |
| B1 = A |
| A = TT1 |
| H = G |
| G = \_rotl(F, 19) |
| F = E |
| E = \_P0(TT2) |
| Vv = [(Vs[0]^A) & 0xFFFFFFFF, (Vs[1]^B1) & 0xFFFFFFFF, (Vs[2]^C) & 0xFFFFFFFF, |
| (Vs[3]^D) & 0xFFFFFFFF, (Vs[4]^E) & 0xFFFFFFFF, (Vs[5]^F) & 0xFFFFFFFF, |
| (Vs[6]^G) & 0xFFFFFFFF, (Vs[7]^H) & 0xFFFFFFFF] |
| res = 0 |
| for x in Vv: |
| res = (res << 32) | (x & 0xFFFFFFFF) |
| return res |
|  |
| def sm3\_hash(msg: bytes) -> bytes: |
| IV = bytes.fromhex('7380166f4914b2b9172442d7da8a0600a96f30bc163138aae38dee4db0fb0e4e') |
| bit\_len = len(msg) \* 8 |
| msg\_padded = msg + b'\x80' |
| k = (56 - (len(msg\_padded) % 64)) % 64 |
| msg\_padded += b'\x00' \* k |
| msg\_padded += struct.pack('>Q', bit\_len) |
| V = int.from\_bytes(IV, 'big') |
| for i in range(0, len(msg\_padded), 64): |
| block = msg\_padded[i:i+64] |
| V = sm3\_compress(V, block) |
| return int\_to\_bytes(V, 32) |
|  |
| # KDF |
| def kdf(z: bytes, klen: int) -> bytes: |
| ct = 1 |
| out = bytearray() |
| while len(out) < klen: |
| out.extend(sm3\_hash(z + struct.pack('>I', ct))) |
| ct += 1 |
| return bytes(out[:klen]) |
|  |
| # SM2 加解密 |
| def generate\_keypair() -> Tuple[int, Tuple[int, int]]: |
| while True: |
| d = bytes\_to\_int(os.urandom(32)) % n |
| if 1 <= d < n: |
| break |
| P = scalar\_mult(d, (Gx, Gy)) |
| return d, P |
|  |
| def sm2\_encrypt(pub: Tuple[int, int], msg: bytes) -> bytes: |
| mlen = len(msg) |
| if mlen == 0: |
| raise ValueError("不支持空消息") |
| while True: |
| k = bytes\_to\_int(os.urandom(32)) % n |
| if k == 0: |
| continue |
| C1 = scalar\_mult(k, (Gx, Gy)) |
| S = scalar\_mult(k, pub) |
| if S is None: |
| continue |
| x2, y2 = S |
| # 只做一次 int->bytes 拼接，传给 KDF 和 C3 |
| xy\_bytes = concat\_ints\_to\_bytes(x2, y2) |
| t = kdf(xy\_bytes, mlen) |
| if int.from\_bytes(t, 'big') == 0: |
| continue |
| C2 = bytes([m ^ t\_i for m, t\_i in zip(msg, t)]) |
| C3 = sm3\_hash(xy\_bytes + msg) |
| return point\_to\_bytes(C1) + C3 + C2 |
|  |
| def sm2\_decrypt(priv: int, C: bytes) -> bytes: |
| if len(C) < 97: |
| raise ValueError("密文长度不合法") |
| C1\_bytes = C[:65] |
| C3 = C[65:97] |
| C2 = C[97:] |
| C1 = bytes\_to\_point(C1\_bytes) |
| S = scalar\_mult(priv, C1) |
| if S is None: |
| raise ValueError("S 为无穷点") |
| x2, y2 = S |
| xy\_bytes = concat\_ints\_to\_bytes(x2, y2) |
| t = kdf(xy\_bytes, len(C2)) |
| if int.from\_bytes(t, 'big') == 0: |
| raise ValueError("KDF 输出全 0") |
| M = bytes([c ^ t\_i for c, t\_i in zip(C2, t)]) |
| if sm3\_hash(xy\_bytes + M) != C3: |
| raise ValueError("C3 校验失败") |
| return M |
|  |
|  |
| # 测试 |
| def bench(rounds: int = 20): |
| print("生成密钥对...") |
| d, P = generate\_keypair() |
|  |
| message = "abc".encode("utf-8") |
| print("原文:", message) |
|  |
| C = sm2\_encrypt(P, message) |
| M = sm2\_decrypt(d, C) |
| print("解密结果:", M) |
| print("解密正确:", M == message) |
|  |
| # 加密 |
| t0 = time.perf\_counter() |
| for \_ in range(rounds): |
| sm2\_encrypt(P, message) |
| t1 = time.perf\_counter() |
| print(f"平均加密时间: {(t1 - t0) / rounds \* 1000:.3f} ms") |
|  |
| # 解密 |
| t0 = time.perf\_counter() |
| for \_ in range(rounds): |
| sm2\_decrypt(d, C) |
| t1 = time.perf\_counter() |
| print(f"平均解密时间: {(t1 - t0) / rounds \* 1000:.3f} ms") |
|  |
| if \_\_name\_\_ == "\_\_main\_\_": |
| bench(10) |

1. 运行结果比较



