# SM2 数字签名算法实验报告

SM2 是中国密码管理局发布的椭圆曲线公钥密码标准算法，包含数字签名、密钥交换和公钥加密功能。本实验聚焦其数字签名功能，完整实现了密钥生成、签名生成和签名验证流程。

**参数选择：**

- SM2 system parameters:
  - $\mathbb{F}_q$ : finite field where $|\mathbb{F}_q| = q$
  - $a, b$: elliptic curve equation parameters
  - $G = (x_G, y_G)$: base point
  - $n$: order
  - $h$: cofactor where $h = \left|E\left(F_q\right)\right|/n$

- SM2 system parameters: prime field
  - Elliptic curve equation: $y^2 = x^3 + ax + b$ over $\mathbb{F}_q$-256
  - Prime $q$: 8542D69E 4C044F18 E8B92435 BF6FF7DE 45728391 5C45517D 722EDB8B 08F1DFC3
  - $a$: 787968B4 FA32C3FD 2417842E 73BBFEFF 2F3C848B 6831D7E0 EC65228B 3937E498
  - $b$: 63E4C6D3 B23B0C84 9CF84241 484BFE48 F61D59A5 B16BA06E 6E12D1DA 27C5249A
  - $G = (x_G, y_G)$, $ord(G) = n$
  - $x_G$: 421DEBD6 1B62EAB6 746434EB C3CC315E 32220B3B ADD50BDC 4C4E6C14 7FEDD43D
  - $y_G$: 0680512B CBB42C07 D47349D2 153B70C4 E5D7FDFC BFA36EA1 A85841B9 E46E09A2
  - $n$ : 8542D69E 4C044F18 E8B92435 BF6FF7DD 29772063 0485628D 5AE74EE7 C32E79B7

**椭圆曲线基础运算**

```
def modular_inverse(value, modulus):
    """计算模逆元（扩展欧几里得算法）"""
    mult_low, mult_high = 1, 0
    val_low, val_high = value % modulus, modulus
    while val_low > 1:
        ratio = val_high // val_low
        new_mult = mult_high - mult_low * ratio
        new_val = val_high - val_low * ratio
        mult_low, val_low, mult_high, val_high = new_mult, new_val, mult_low, val_low
    return mult_low % modulus
```

计算有限域内的模逆元，使用扩展欧几里得算法实现：初始化乘数因子和余数，通过迭代计算余数和乘数，当余数减至 1 时返回乘数因子

**椭圆曲线点运算**

```
def ecc_point_addition(point1, point2):
    """椭圆曲线两点加法"""
    slope = ((point2[1] - point1[1]) *
             modular_inverse(point2[0] - point1[0], MODULUS_P)) % MODULUS_P
    result_x = (slope * slope - point1[0] - point2[0]) % MODULUS_P
    result_y = (slope * (point1[0] - result_x) - point1[1]) % MODULUS_P
    return (result_x, result_y)
```

算法原理:

计算通过两点的斜率 λ

利用斜率求新点 x 坐标：$x_3 = \lambda^2 - x_1 - x_2$

计算新点 y 坐标：$y_3 = \lambda(x_1 - x_3) - y_1$

```
def ecc_point_doubling(point):
    slope = ((3 * point[0] * point[0] + ECC_PARAM_A) *
             modular_inverse((2 * point[1]), MODULUS_P)) % MODULUS_P
    result_x = (slope * slope - 2 * point[0]) % MODULUS_P
    result_y = (slope * (point[0] - result_x) - point[1]) % MODULUS_P
    return (result_x, result_y)
```

点加自身时斜率为$(3x_1^2 + a)/2y_1$，其他计算类似点加法

**标量乘法**

```
def scalar_point_multiplication(scalar, base_point):
    """标量乘法（倍点运算）"""
    if scalar == 0 or scalar >= ORDER_N:
        raise ValueError("Invalid scalar or private key")

    scalar_binary = bin(scalar)[2:]
    temp_point = base_point
```

```
        for bit in scalar_binary[1:]:
            temp_point = ecc_point_doubling(temp_point)
            if bit == "1":
                temp_point = ecc_point_addition(temp_point, base_point)

        return temp_point
```

**实现策略**:

1. 将标量转换为二进制格式
2. 使用双倍-加法优化算法
3. 首位后处理：每次双倍后根据二进制位决定是否加点
4. 安全验证：检查标量范围（0 < k < n）

**用户标识哈希计算**

```
def compute_identity_hash(user_id, pub_x, pub_y):
    """计算用户身份哈希值 Z_A"""
    components = [
        str(calculate_bit_length(user_id)),
        user_id,
        str(ECC_PARAM_A),
        str(ECC_PARAM_B),
        str(BASE_PT_X),
        str(BASE_PT_Y),
        str(pub_x),
        str(pub_y)
    ]
    concatenated = "".join(components)
    digest = sm3.sm3_hash(func.bytes_to_list(concatenated.encode()))
```

```
    return int(digest, 16)
```

1. 按标准要求拼接参数：ID 长度、曲线参数、基点、公钥
2. 使用 SM3 哈希算法处理拼接后的字符串
3. 转换为整数值用于后续签名

**密钥对生成**

```
def create_key_pair():
    """生成 SM2 密钥对"""
    private_val = int(secrets.token_hex(32), 16) % ORDER_N
    public_pt = scalar_point_multiplication(private_val, BASE_POINT)
    return private_val, public_pt
```

使用 secrets 模块生成密码学安全的随机数

限制私钥范围在[1, n-1]之间

通过标量乘法从私钥导出公钥

**签名生成算法**

```
def generate_signature(private_key, msg_content, user_hash):
    """生成数字签名"""
    merged_data = str(user_hash) + msg_content
    msg_bytes = merged_data.encode()
    digest_hash = sm3.sm3_hash(func.bytes_to_list(msg_bytes))
    hash_int = int(digest_hash, 16)

    k_input = str(private_key) + sm3.sm3_hash(func.bytes_to_list(msg_content.encode()))
    k_val = int(sha256(k_input.encode()).hexdigest(), 16)

    if k_val >= MODULUS_P:
        return None
```

```
    temp_pt = scalar_point_multiplication(k_val, BASE_POINT)
    r_val = (hash_int + temp_pt[0]) % ORDER_N
    s_val = modular_inverse(1 + private_key, ORDER_N) * (k_val - r_val * private_key) %
ORDER_N
    return (r_val, s_val)
```

1. 组合用户哈希和消息，计算 SM3 哈希 e
2. 使用 RFC6979 方法生成确定性的 k 值
3. 计算椭圆曲线点$(k \cdot G) = (x_1, y_1)$
4. 生成签名分量 $r = (e + x_1) \bmod n$
5. 计算签名分量 $s = ((1 + d)^{-1} \cdot (k - r \cdot d)) \bmod n$

## 签名验证算法

```
def verify_signature(public_key, user_id, msg_content, signature):
    """验证数字签名"""
    r_val, s_val = signature
    user_hash = compute_identity_hash(user_id, public_key[0], public_key[1])
    merged_data = str(user_hash) + msg_content
    msg_bytes = merged_data.encode()
    digest_hash = sm3.sm3_hash(func.bytes_to_list(msg_bytes))
    hash_int = int(digest_hash, 16)

    composite_val = (r_val + s_val) % ORDER_N
    point1 = scalar_point_multiplication(s_val, BASE_POINT)
    point2 = scalar_point_multiplication(composite_val, public_key)
    result_pt = ecc_point_addition(point1, point2)

    R_val = (hash_int + result_pt[0]) % ORDER_N
    return R_val == r_val
```

1. 重新计算用户标识哈希 Z_A
2. 组合 Z_A 和消息计算哈希值 e'
3. 计算 t = (r + s) mod n
4. 计算椭圆曲线点：s·G + t·P_A
5. 比对 R' = (e' + $x_1$') mod n 与 r 是否相等

**测试**

```
=== 测试案例 #1 ===
用户ID: user_12345
消息内容: 重要的数字证书申请
生成的公钥: (0x18782aefe6515080bd4e1d7a3a15d87990362953a4d035f410270cd24593fb24, 0x48a66c155b0bb115111d89e998550a8420818b6f4f0e5b87b7378a68becc7ef9)
签名结果: r=0x72dcad7207cadb14ddcc823d9b6b95e4575730c81bd8e423fac65953040d930a, s=0x75faeb7fedc51e9b7e4894eb1f7fc4dbfd5d84f4ea55ac3c1c580da3bbac369e
签名验证状态: 成功

=== 测试案例 #2 ===
用户ID: admin@org.com
消息内容: 系统配置文件备份
生成的公钥: (0x2330335948cf4259c0565f159ed24b926b6f7df7381e1cc0e86cab549c6d9166, 0x15e03e143f6df3dd384ed493953b91c5d3f82910726eaa74c1037ff65ae0e663)
签名结果: r=0x5de946b7df082273e4527b236a8e4ef6ef8157dc6801427d1abddb25d5ade190, s=0x3425feabd7ab0aad3d2427a72d8bcea52b1e4c7055a771d4392380a0dee688da
签名验证状态: 成功

=== 测试案例 #3 ===
用户ID: device_abc
消息内容: 固件升级请求
生成的公钥: (0x19c9744bd8494421bce2c118909327308b470f17582a7c0a24a9dc3d8a738893, 0x517941d54089a567d6356f33eedbe9df3063ec1079dacc719bcf688b2bbc6f0d)
签名结果: r=0x59288656f411a144abad517bedffbd54d6ae9409b3c7e4f8ce84df9e751df17c, s=0x64282bac7f3549dc6d167915dd7460cca1ca81d1db2f949fe9996d9b50ddfd51
签名验证状态: 成功
```

**基本实现了 sm2 的基础功能**