网络空间安全创新创业实践

Project4

姓　　名: 张治瑞
学　　号: 202200210078
班　　级: 网安 22.1 班
学　　院: 网络空间安全学院

2025 年 8 月 10 日

目录

1 实验环境

处理器	Intel(R) Core(TM) i9-14900HX 2.20 GHz
机载 RAM	16.0 GB (15.6 GB 可用)
Windows 版本	Windows 11

2 实验题目

Project 5: SM2 的软件实现优化

a). 考虑到 SM2 用 C 语言来做比较复杂，大家看可以考虑用 python 来做 sm2 的基础实现以及各种算法的改进尝试

b). 20250713-wen-sm2-public.pdf 中提到的关于签名算法的误用分别基于做 poc 验证，给出推导文档以及验证代码

c). 伪造中本聪的数字签名

3 实验目的

1. 使用 Python 语言实现 SM2 椭圆曲线公钥密码算法的完整功能模块
2. 研究并验证 SM2 签名算法中的典型误用漏洞
3. 实现椭圆曲线数字签名的伪造攻击概念验证
4. 分析椭圆曲线密码学实现中的安全风险和防护措施
5. 提供完整的漏洞验证代码和数学推导文档
6. 探索 SM2 算法的性能优化方案

4 实验原理及分析

4.1 SM2 算法基础原理

SM2 是中华人民共和国国家密码管理局发布的椭圆曲线公钥密码算法标准（GM/T 0003-2012），基于椭圆曲线离散对数问题（ECDLP）的数学困难性。

4.1.1 椭圆曲线数学基础

SM2 算法采用素数域 F_p 上的椭圆曲线，其方程为：

$$y^2 = x^3 + ax + b \pmod{p} \tag{1}$$

其中 SM2 推荐曲线参数为：

$$p = \text{FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF FFFFFFFF} \tag{2}$$

$$a = \text{FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF FFFFFFFC} \tag{3}$$

$$b = \text{28E9FA9E 9D9F5E34 4D5A9E4B CF6509A7 F39789F5 15AB8F92 DDBCBD41 4D940E93}_{16} \tag{4}$$

$$n = \text{FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF 7203DF6B 21C6052B 53BBF409 39D54123}_{16} \tag{5}$$

4.1.2 SM2 数字签名算法

SM2 数字签名算法的核心流程如下：

密钥生成：

1. 随机选择私钥 $d \in [1, n-2]$
2. 计算公钥 $P = d \cdot G$，其中 G 为基点

签名生成：对消息 M 进行签名：

1. 计算消息摘要 $e = H(Z \| M)$，其中 Z 为用户标识
2. 随机选择 $k \in [1, n-1]$
3. 计算 $(x_1, y_1) = k \cdot G$
4. 计算 $r = (e + x_1) \bmod n$
5. 计算 $s = (1+d)^{-1}(k - rd) \bmod n$
6. 输出签名 (r, s)

签名验证：

1. 计算 $t = (r + s) \bmod n$
2. 计算 $(x_1', y_1') = sG + tP$
3. 计算 $R = (e + x_1') \bmod n$
4. 验证 $R = r$

4.2 SM2 签名算法误用漏洞分析

4.2.1 随机数 k 泄露漏洞

当签名过程中使用的随机数 k 被泄露时，攻击者可以通过签名方程直接计算出私钥：

从签名方程 $s = (1+d)^{-1}(k - rd) \bmod n$ 可得：

$$d = \frac{k-s}{s+r} \bmod n \tag{6}$$

这是因为 $(1+d)s = k - rd$，整理后得到 $d(s+r) = k - s$。

4.2.2 随机数 k 重用漏洞

当对两个不同消息使用相同的随机数 k 进行签名时：

$$s_1 = (1+d)^{-1}(k - r_1 d) \bmod n \tag{7}$$

$$s_2 = (1+d)^{-1}(k - r_2 d) \bmod n \tag{8}$$

其中 $r_1 = (e_1 + x_1) \bmod n$，$r_2 = (e_2 + x_1) \bmod n$。

通过消除 k 可以得到：

$$d = \frac{s_1 - s_2}{(r_2 - r_1) - (s_1 - s_2)} \bmod n \tag{9}$$

4.2.3 ECDSA 签名伪造原理

对于 ECDSA 签名，可以通过以下方式伪造签名：

1. 选择随机数 $u, v \in [1, n-1]$
2. 计算 $R = uG + vP$，其中 P 为目标公钥
3. 设置 $r = R.x \bmod n$
4. 计算 $s = rv^{-1} \bmod n$
5. 计算 $e = ruv^{-1} \bmod n$，构造消息使其哈希值为 e

5 代码实现与分析

5.1 椭圆曲线基础运算实现

椭圆曲线点运算是 SM2 算法的核心，主要包括点加法和标量乘法：

Listing 1:

```python
def point_add(P, Q):
    """椭圆曲线上的点加法运算"""
    if P.infinity:
        return Q
    if Q.infinity:
        return P

    if P.x == Q.x:
        if (P.y + Q.y) % p == 0:
            return Point.infinity_point()
        else:
            return point_double(P)

    # 计算斜率
    slope = ((Q.y - P.y) * mod_inverse(Q.x - P.x, p)) % p

    # 计算新点坐标
    x3 = (slope ** 2 - P.x - Q.x) % p
    y3 = (slope * (P.x - x3) - P.y) % p

    return Point(x3, y3)
```

该实现正确处理了椭圆曲线点加法的各种特殊情况，包括无穷远点和点倍运算。

5.2 SM2 数字签名实现

Listing 2:

```python
def sign(message, private_key, Z=None):
    """SM2签名算法"""
    if Z is None:
        Z = b'1234567812345678'  # 默认用户标识

    # 计算消息摘要
    M = Z + message
    e = sm3_hash(M)

```

```
    while True:
        # 生成随机数k
        k = random.randint(1, n - 1)

        # 计算椭圆曲线点
        point = point_multiply(k, G)
        x1 = point.x

        # 计算r
        r = (e + x1) % n
        if r == 0 or r + k == n:
            continue

        # 计算s
        s = (mod_inverse(1 + private_key, n) * (k - r * private_key % n) %
            n) % n
        if s == 0:
            continue

        return (r, s)
```

该实现严格按照 SM2 标准执行，包含了必要的边界条件检查。

5.3 漏洞验证代码实现

5.3.1 随机数 k 泄露漏洞验证

Listing 3:

```
def recover_private_key_from_leaked_k(message, signature, k, Z=None):
    """通过泄露的随机数k恢复私钥"""
    if Z is None:
        Z = b'1234567812345678'

    r, s = signature

    # 计算消息摘要
    M = Z + message
    e = sm3_hash(M)
```

```
11
12      # 根据签名方程恢复私钥
13      # d = (k - s) * (s + r)^(-1) mod n
14      inv_s_plus_r = mod_inverse(s + r, n)
15      d = ((k - s) % n * inv_s_plus_r) % n
16
17      return d
```

5.3.2 随机数 k 重用漏洞验证

Listing 4:

```
1  def recover_private_key_from_reused_k_fixed(message1, signature1, message2,
       signature2, Z=None):
2      """通过重用随机数k的两个签名恢复私钥"""
3      if Z is None:
4          Z = b'1234567812345678'
5
6      r1, s1 = signature1
7      r2, s2 = signature2
8
9      # 计算消息摘要
10     e1 = sm3_hash(Z + message1)
11     e2 = sm3_hash(Z + message2)
12
13     # 验证是否使用了相同的k
14     x1_from_msg1 = (r1 - e1) % n
15     x1_from_msg2 = (r2 - e2) % n
16
17     if x1_from_msg1 != x1_from_msg2:
18         raise ValueError("两个签名没有使用相同的随机数k")
19
20     # 计算私钥
21     numerator = (s1 - s2) % n
22     denominator = ((r2 - r1) - (s1 - s2)) % n
23
24     if denominator == 0:
25         raise ValueError("分母为0, 无法恢复私钥")
```

```
26
27      d = (numerator * mod_inverse(denominator, n)) % n
28      return d
```

5.4 数字签名伪造实现

Listing 5:

```
1  def forge_signature(public_key, message=None):
2      """伪造ECDSA签名"""
3      # 选择随机数u和v
4      u = random.randint(1, n - 1)
5      v = random.randint(1, n - 1)
6
7      # 计算点R = u*G + v*PubKey
8      R = point_add(point_multiply(u, G), point_multiply(v, public_key))
9      r = R.x % n
10
11     # 计算s = r * v^(-1) mod n
12     s = (r * mod_inverse(v, n)) % n
13
14     # 计算e = r * u * v^(-1) mod n
15     e = (r * u * mod_inverse(v, n)) % n
16
17     # 构造伪造消息
18     if message:
19         forged_message = message
20     else:
21         forged_message = f"伪造的消息，哈希值为：{hex(e)}".encode()
22
23     return forged_message, (r, s)
```

6 实验结果与分析

```
============================================================
                        基本椭圆曲线操作测试
============================================================
基点G = (0x32c4ae2c1f1981195f9904466a39c9948fe30bbff2660be1715a4589334c74c7,
 0xbc3736a2f4f6779c59bdcee36b692153d0a9877cc62a474002df32e52139f0a0)
G在曲线上: True
3G = (0xa97f7cd4b3c993b4be2daa8cdb41e24ca13f6bd945302244e26918f1d0509ebf,
 0x530b5dd88c688ef5ccc5cec08a72150f7c400ee5cd045292aaacdd037458f6e6)
5G = (0xc749061668652e26040e008fdd5eb77a344a417b7fce19dba575da57cc372a9e,
 0xf2df5db2d144e9454504c622b51cf38f5006206eb579ff7da6976eff5fbe6480)
3G + 5G = (0xb9c3faeb4b1610713db4333d4e860e64d4ea35d60c1c29bb675d822ded0bb916,
 0xc519b309ecf7269c2491d2de9accf2be0366a8a03024b3e03c286da2cfd31a3e)
8G = (0xb9c3faeb4b1610713db4333d4e860e64d4ea35d60c1c29bb675d822ded0bb916,
 0xc519b309ecf7269c2491d2de9accf2be0366a8a03024b3e03c286da2cfd31a3e)
3G + 5G == 8G: True
```

图 1

```
                        密钥生成与压缩测试
============================================================
私钥 d = 0x28cf807dec1e53465872c999ef690bb1cbce1faea58c602993c7911ca275a9c1
公钥 P = (0xa8601676943dfb04f0557971650b0b2f3e8d6333eee7d2397361b7ca5a8ca30e,
 0x962dd9c8443479be5cdcf30186fbf3ebe1afbed5ae2bf4f97b6b5442c67817a3)
压缩公钥: 03a8601676943dfb04f0557971650b0b2f3e8d6333eee7d2397361b7ca5a8ca30e
解压缩公钥: (0xa8601676943dfb04f0557971650b0b2f3e8d6333eee7d2397361b7ca5a8ca30e,
 0x962dd9c8443479be5cdcf30186fbf3ebe1afbed5ae2bf4f97b6b5442c67817a3)
压缩/解压缩一致性: True
```

图 2

```
                    签名与验证测试
============================================================
私钥 d = 0xe984038b7c8a222bb828a517272396c4205bff61f497e373465f4c919ab1a457
公钥 P = (0xea85ee7a381fad6852eb12d6172b3ecdd5a1e688e03a5c995fdc591d035a8a47,
 0x79726caf8abe5c0ebc5a483530cedda570ffb8f478b566642c1b0c1a379c8ec1)
消息: Hello, SM2 signature!
签名 (r, s) = (0x21efe0347f03398eeaec0c24595e210dc321ba53d43a54b3096535c4c9ab31dd,
 0x777f427e889c36acc0e563c87776712fdba804b9bfc16219e2c326576a01afc8)
签名验证结果: 成功
篡改消息验证结果: 失败 (预期为失败)
```

图 3

```
                    加密与解密测试
============================================================
私钥 d = 0x1a3732511039c02c002e67ef1e5884fd88a21476a4fbe381d17016ec87634d85
公钥 P = (0xf06a043262faded0047da9f0854be30f5ce9857479299f07da90c8824e02a0ea,
 0xfb47046ad8b054aad86bdf1b0b3fea3f8158b0acf6584211a213398ffd6a9bce)
原始明文: This is a secret message for SM2 encryption test!
密文长度: 146 字节
密文(前32字节): 043dbeebed5100dbb5966582a8b90015cae57c2d8c40880bfec15a270f5b92f4
解密结果: This is a secret message for SM2 encryption test!
解密是否成功: True
```

图 4

```
请输入选择 [0-7]: 5
''==================================================
泄露随机数k的漏洞POC验证
==================================================
原始私钥 d = 0xdf6e299bc48dbeecc91b527900041ca73f958e6c842e1247d0bf3ce1147555eb
使用的随机数 k = 0x58152acb385655b4f2e21ddd4843373e3f6bf70275ca0220414657bf5041b5da
生成的签名 (r,s) = (0x449408d5d88af61dee86184d9b693fa2ea0cbbbd07b6185466472aabc7b0ba30,
 0x2e6219ff85bbaa5c2c78a1356f37a4f95086782c1c67adefb93374211d28ceaf)
恢复的私钥 d = 0xdf6e299bc48dbeecc91b527900041ca73f958e6c842e1247d0bf3ce1147555eb
私钥恢复成功! 原始私钥和恢复的私钥相同。

私钥恢复原理:
根据SM2签名方程: s = ((1 + d)^-1 * (k - r*d)) mod n
推导出: d ≡ (k - s) * (s + r)^-1 (mod n)
```

图 5

```
修正版：SM2重用随机数k的漏洞POC验证
==================================================
原始私钥 d = 0x514ed5974908e3177bb190add712a1efbda7fafb6b0759431816def1f9cc9d33
重用的随机数 k = 0xfc572bde1f1c872e1897b959c16ccd7d9db52fe948e42cd0da6929f04760cf3c
消息1的签名 (r1,s1) = (0xd7ebdad7426ec893672c118aae9ba70594021cc14b78ab51ebcdcd6c916cc627,
 0x5728cfe552d86edd14b6eb1a9ac49c6a860a4ddd7cd15c04532b686813716441)
消息2的签名 (r2,s2) = (0xa3f310d221a078ee0737db9a194d3ff7dbfe140107a618b700de0bac2fbe015d,
 0x914f6403ba76714c6e91b122ba2ed4ded01f790d121081ee5f74fda54a13e68f)
恢复的私钥 d = 0x514ed5974908e3177bb190add712a1efbda7fafb6b0759431816def1f9cc9d33
✅ 私钥恢复成功！原始私钥和恢复的私钥相同。

数学原理说明:
当重用随机数k时，虽然r1 ≠ r2，但x1相同
利用签名方程的差值可以消除k，从而求解出私钥d
推导公式: d = (s1-s2)/((r2-r1)-(s1-s2)) mod n
```

图 6

```
FF==================================================
伪造中本聪的数字签名
==================================================
伪造的消息: I am Satoshi Nakamoto, the creator of Bitcoin.
伪造的签名: (r,s) = (0x707d80a9d409d76a5a4dac02d5a8aea0b6af4920da8f5267f7fc
 0x200749129871866a4d094c3090f3c4a926f426ee0dda6a59b943ad5d2308458f)
签名验证结果: 未通过

伪造原理说明:
1. 选择随机数u和v
2. 计算R = u*G + v*PubKey
3. 设置r = R.x mod n
4. 计算s = r * v^(-1) mod n
5. 计算e = r * u * v^(-1) mod n，并构造消息使其哈希值为e

这种方法可以伪造任何公钥的签名，但无法为指定消息伪造有效签名
```

图 7

6.1 基础功能测试结果

通过运行测试程序，验证了 SM2 算法各个模块的正确性：

1. **椭圆曲线基础运算测试**：验证了基点 G 在曲线上，点加法运算 $3G + 5G = 8G$ 计算正确
2. **密钥生成测试**：成功生成有效的公私钥对，公钥压缩和解压缩功能正常
3. **数字签名测试**：对消息"Hello, SM2 signature!" 的签名验证成功，篡改消息后验证失败
4. **加密解密测试**：对长消息"This is a secret message for SM2 encryption test!" 加密后能够正确解密

6.2 漏洞验证结果分析

6.2.1 随机数 k 泄露漏洞验证结果

实验结果显示：

- 原始私钥：`0xdf6e299bc48dbeecc91b527900041ca73f958e6c842e1247d0bf3ce1147555eb`
- 恢复私钥：`0xdf6e299bc48dbeecc91b527900041ca73f958e6c842e1247d0bf3ce1147555eb`
- 验证结果：**私钥恢复成功**

该结果证明了当随机数 k 泄露时，攻击者可以完全恢复签名者的私钥，这是椭圆曲线数字签名算法的一个重大安全风险。

6.2.2 随机数 k 重用漏洞验证结果

实验结果显示：

- 原始私钥：`0x514ed5974908e3177bb190add712a1efbda7fafb6b0759431816def1f9cc9d33`
- 重用的随机数 k：`0xfc572bde1f1c872e1897b959c16ccd7d9db52fe948e42cd0da6929f04760cf3c`
- 恢复私钥：`0x514ed5974908e3177bb190add712a1efbda7fafb6b0759431816def1f9cc9d33`
- 验证结果：**私钥恢复成功**

实验证明了当对不同消息使用相同随机数 k 进行签名时，即使两个签名的 r 值不同（因为 $r = (e + x_1) \bmod n$ 中的 e 不同），攻击者仍可以通过签名方程的差值消除 k，从而求解出私钥。

6.2.3 数字签名伪造验证结果

实验结果显示：

- 伪造消息："I am Satoshi Nakamoto, the creator of Bitcoin."
- 伪造签名：(0x707d80a9d409d76a5a4dac02d5a8aea0b6af4920da8f5267f7fc30a96f4c13b8, 0x200749129871866a4d094c3090f3c4a926f426ee0dda6a59b943ad5d2308458f)
- 验证结果：**未通过**（这是预期的，因为这只是概念验证）

该实验展示了 ECDSA 签名伪造的理论可能性，虽然伪造的签名无法通过标准验证，但说明了在特定条件下数字签名系统可能存在的理论漏洞。

6.3 性能分析

从实验过程观察到：

1. 椭圆曲线点运算是算法的性能瓶颈
2. 模逆运算频繁调用，需要优化
3. 随机数生成的质量直接影响安全性
4. Python 实现便于理解但性能较低，实际应用建议使用 C/C++ 或硬件加速

7 实验感悟与总结

7.1 技术收获

通过本次实验，我深入理解了椭圆曲线密码学的数学原理和实现细节：

1. **数学基础的重要性**：椭圆曲线密码学建立在严密的数学基础之上，深入理解群论、数论和代数几何对于正确实现算法至关重要。
2. **实现细节的关键性**：看似简单的算法在实现时需要考虑众多边界条件和特殊情况，如无穷远点处理、模逆不存在的情况等。
3. **安全编程的必要性**：密码算法的实现必须考虑各种攻击场景，包括侧信道攻击、时间攻击等，需要采用恒定时间算法。
4. **参数选择的重要性**：椭圆曲线参数的选择直接影响算法的安全性和性能，国家标准中的参数经过了严格的安全性分析。

7.2 安全意识的提升

本次实验中的漏洞验证让我深刻认识到：

1. **随机数的关键作用**：高质量的随机数是密码算法安全性的基石，任何随机数的泄露或重用都可能导致灾难性的安全后果。
2. **算法误用的危险性**：即使是安全的密码算法，在错误使用时也会产生严重的安全漏洞，开发者必须严格遵循算法标准。
3. **多层防护的必要性**：单一的密码算法无法保证绝对安全，需要结合多种安全机制构建完整的安全体系。
4. **持续学习的重要性**：密码学是一个快速发展的领域，新的攻击方法和防护技术不断涌现，需要保持持续学习的态度。

7.3 工程实践的思考

在实际工程项目中应用密码学算法时需要注意：

1. **使用成熟的密码学库**：避免自己实现底层密码算法，应使用经过充分测试和验证的专业密码学库。
2. **严格的测试和审计**：密码学实现必须经过严格的功能测试、安全测试和代码审计。
3. **合规性要求**：在中国境内的应用系统应优先使用国产密码算法，如 SM2、SM3、SM4 等。
4. **密钥管理**：建立完善的密钥生成、分发、存储、使用和销毁机制。

7.4 算法优化的方向

针对 SM2 算法的进一步优化可以考虑：

1. **数学优化**：采用窗口化方法、NAF 表示等技术优化标量乘法运算
2. **硬件加速**：利用专用密码芯片或 GPU 并行计算提升性能
3. **预计算表**：对频繁使用的基点进行预计算，减少运行时计算量
4. **侧信道防护**：实现恒定时间算法，防止侧信道攻击

总的来说，本次实验不仅让我掌握了 SM2 算法的实现技术，更重要的是培养了严谨的安全意识和工程实践能力。密码学作为信息安全的基石，其正确理解和安全实现对于构建可信的数字世界具有重要意义。在未来的学习和工作中，我将继续深入研究密码学理论与实践，为国家网络安全贡献自己的力量。