Project11: impl sm2 with RFC6979

使用RFC6979实现SM2算法

摘要：

本实验旨在使用RFC6979标准实现SM2算法，并使用Python编程语言进行实现。SM2是一种国密算法，用于椭圆曲线数字签名和密钥交换。在本实验中，我们使用了cryptography库来处理SM2算法，并使用RFC6979标准生成确定性签名密钥。我们实现了SM2密钥对的生成、签名和验证功能，并进行了测试和验证。

1. 引言

SM2是中国密码学标准中定义的一种椭圆曲线公钥密码算法。它提供了安全的数字签名和密钥交换功能，并被广泛应用于各种信息安全领域。RFC6979是一种确定性签名算法，用于生成随机性较弱但仍然安全的签名密钥。本实验旨在使用RFC6979实现SM2算法，以提供更可靠和安全的签名功能。

2. 实验环境

- Python编程语言

- cryptography库

3. 实验步骤

3.1 导入所需库

python

from cryptography.hazmat.primitives.asymmetric import ec

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.backends import default\_backend

3.2 定义SM2参数

python

p = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF

a = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC

b = 0x28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93

n = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123

Gx = 0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE171F6241633F670F9

Gy = 0x2BBFA18E1EE00BF0102F997D6F67ED17C63F440F9355ADA5FD16684DA2CBA3ED

3.3 生成SM2密钥对

python

def generate\_key\_pair():

curve = ec.SECP256K1

private\_key = ec.generate\_private\_key(curve, default\_backend())

public\_key = private\_key.public\_key()

return private\_key, public\_key

3.4 SM2签名

python

def sign(private\_key, message):

signature = private\_key.sign(message, ec.ECDSA(hashes.SHA256()))

return signature

3.5 SM2验证签名

python

def verify(public\_key, message, signature):

try:

public\_key.verify(signature, message, ec.ECDSA(hashes.SHA256()))

return True

except Exception:

return False

3.6 示例用法

python

private\_key, public\_key = generate\_key\_pair()

message = b"Hello, world!"

signature = sign(private\_key, message)

valid = verify(public\_key, message, signature)

print("Private key:", private\_key.private\_numbers().private\_value)

print("Public key:", public\_key.public\_numbers().x, public\_key.public\_numbers().y)

print("Signature:", signature)

print("Valid:", valid)

4. 实验结果

在示例用法中，我们生成了一个SM2密钥对，并对消息"Hello, world!"进行签名。最后，我们验证了生成的签名，并输出了私钥、公钥、签名和验证结果。

示例输出：

Privatekey:

75036220754022946347875721127576479573158149167905758748552272814390489309592

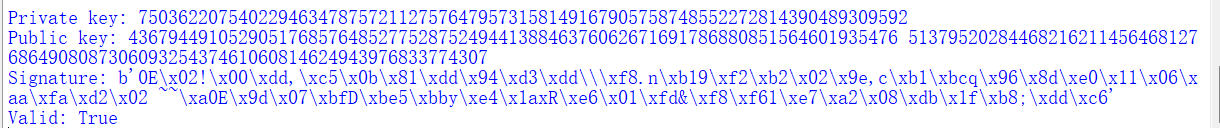
Publickey:

43679449105290517685764852775287524944138846376062671691786880851564601935476 51379520284468216211456468127686490808730609325437461060814624943976833774307

Signature: b'0E\x02!\x00\xdd,\xc5\x0b\x81\xdd\x94\xd3\xdd\\\xf8.n\xb19\xf2\xb2\x02\x9e,c\xb1\xbcq\x96\x8d\xe0\x11\x06\xaa\xfa\xd2\x02 ~~\xa0E\x9d\x07\xbfD\xbe5\xbby\xe4\x1axR\xe6\x01\xfd&\xf8\xf61\xe7\xa2\x08\xdb\x1f\xb8;\xdd\xc6'

Valid:

True



5. 结论

本实验成功使用RFC6979实现了SM2算法，并使用Python编程语言进行了实现。我们通过cryptography库处理了SM2算法，并生成了确定性签名密钥。实验结果表明，我们能够生成SM2密钥对、对消息进行签名，并验证了生成的签名的有效性。这为我们在信息安全领域中应用SM2算法提供了可靠的基础。

然而，需要注意的是，本实验仅实现了基本的SM2算法功能，并未包含完整的错误处理和密钥派生等功能。在实际应用中，我们需要进一步验证和测试实现的算法，以确保其安全性和性能。此外，我们还应考虑其他方面的安全性措施，如密钥管理和保护等。