## Project-21，Schnorr Bacth

### 1，实验内容

收集待验证的签名和相关信息：

获得要验证的一系列Schnorr签名（每个签名包括公钥、消息、签名本身）。

收集对应的公钥。

计算批量验证的挑战值（Challenge Calculation）：

将收集到的每个签名的公钥串联起来，并对其进行哈希运算，生成挑战值。

确保使用相同的哈希算法和消息数据对每个公钥进行串联和哈希运算。

计算批量验证的伴随元素（Accumulated Point Calculation）：

对收集到的每个签名的公钥进行标量乘法运算，使用挑战值作为标量。

将每次乘法运算的结果累加，得到一个累积点（accumulated point）。

进行批量验证：

对收集到的每个签名的签名本身进行标量乘法运算，同样使用挑战值作为标量。

将每次乘法运算的结果累加，得到一个累积签名（accumulated signature）。

验证累积签名：

计算累积签名的验证方程式，检查是否满足等式：[累积签名] = [挑战值] × [累积点]。

如果验证方程式成立，则表明所有签名均有效；如果不成立，则至少有一个签名无效。

实现Schnorr签名的批量验证

(C:\Users\Mr.smile\AppData\Roaming\marktext\images\2023-08-02-17-28-18-image.png)

Schnorr signature batch verify的原理如下：

![](C:\Users\Mr.smile\AppData\Roaming\marktext\images\2023-08-02-17-29-07-image.png)

代码实现中，batch verify部分的实现：

```python

def verify\_sig(sig,M,P):

s\_sum=0

R\_sum=None

eiPi\_sum=None

for i in range(0,Index):

s\_sum+=sig[i][1]

R\_sum=point\_add(R\_sum,sig[i][0])

eiPi\_sum=point\_add(eiPi\_sum,point\_mul(P[i],e[i]))

sG=point\_mul(G,s\_sum)

ReP=point\_add(R\_sum,eiPi\_sum)

if(sG==ReP):

print('签名验证成功')

else:

print('签名验证失败')

```

### 2，运行指导

直接运行代码即可，需要给出消息，默认的Index=4，需要输入四条消息，然后批量验证后输出测试时间。

### 3，运行结果

运行代码结果如下：

> \*\*Index=4\*\* :

>

> \* Batch verify：

>

> ![](C:\Users\Mr.smile\AppData\Roaming\marktext\images\2023-08-02-17-32-45-image.png)

> \*\*Index=4\*\* :

>

> \* single verify time\*Index:

>

> ![](C:\Users\Mr.smile\AppData\Roaming\marktext\images\2023-08-02-17-35-27-image.png)

> \*\*Index=8\*\* :

>

> \* Batch verify:

>

> ![](C:\Users\Mr.smile\AppData\Roaming\marktext\images\2023-08-02-17-36-35-image.png)

> \*\*Index=8\*\* :

>

> \* single verify time\*Index:

>

> ![](C:\Users\Mr.smile\AppData\Roaming\marktext\images\2023-08-02-17-37-40-image.png)

上述通过比较普通验签和批量验签的效率，可以看出批量验证的效率要高于单次的验签。

批量验证在在区块链中，每个区块通常包含多个交易，并需要验证每个交易的签名。通过使用批量验证技术，可以在处理区块时同时验证所有交易的签名，提高整个区块链的吞吐量和效率。

综上，batch verify可以提高验签效率。