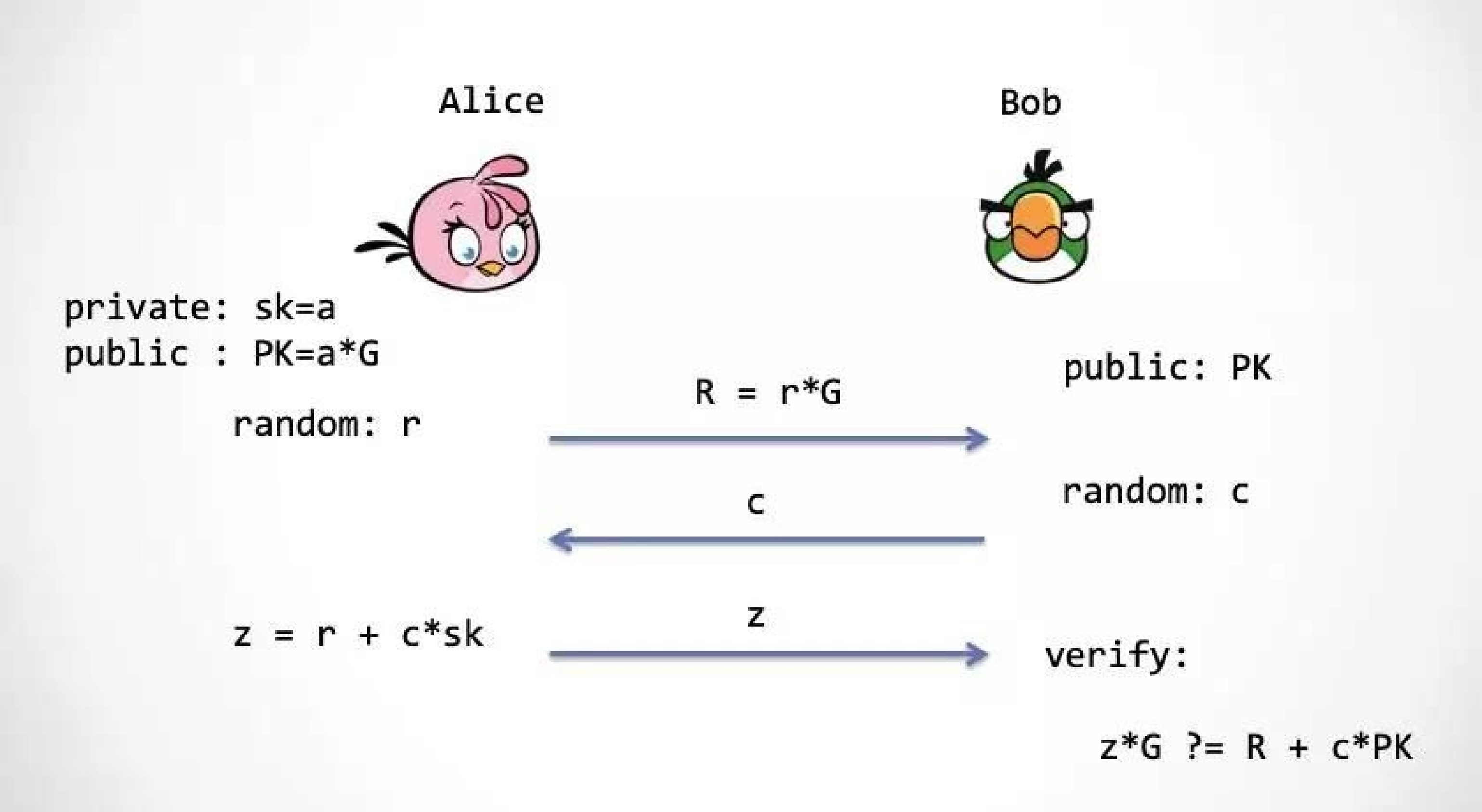
数学基础（schnorr签名）

椭圆曲线加密schnorr机制-零知识

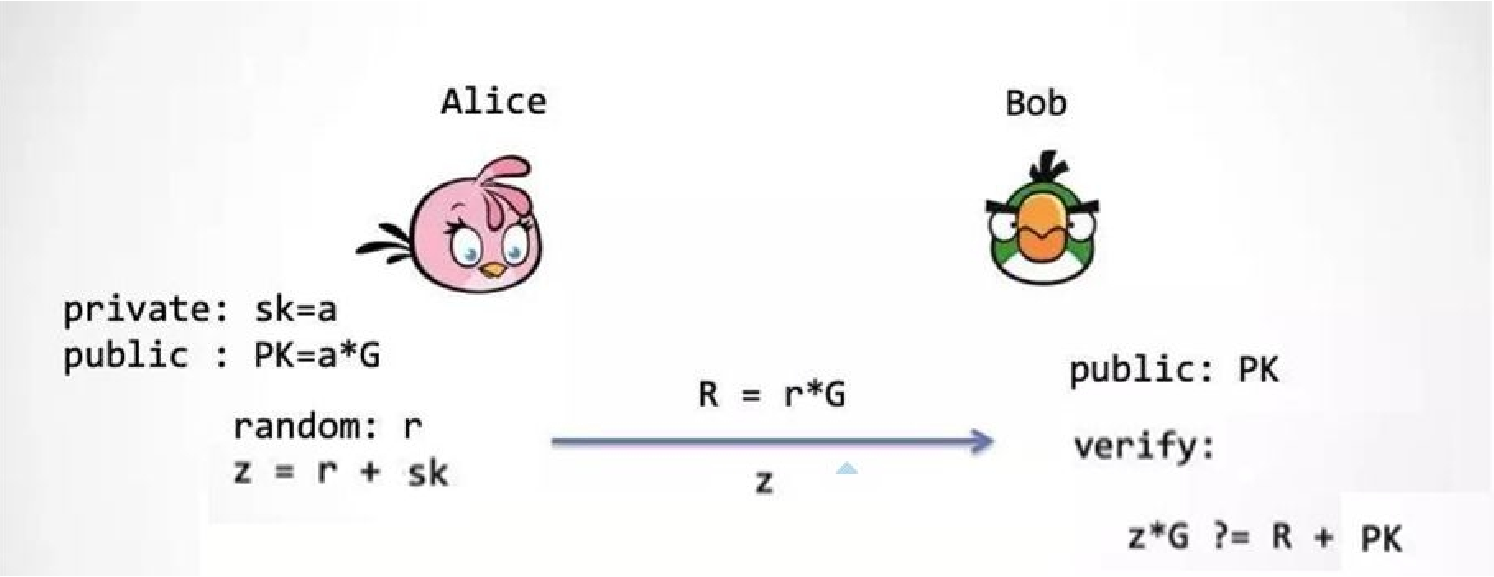
交互式Schnorr协议流程

由于z=r+c*sk，等式两边同时添加相同的生成元可得：z*G=c*PK+R。即可验证Alice确实拥有私钥sk，但是验证者Bob并不能得到私钥sk的值，因此这个过程是零知识的，且是交互式的。此协议也叫做Sigma protocol



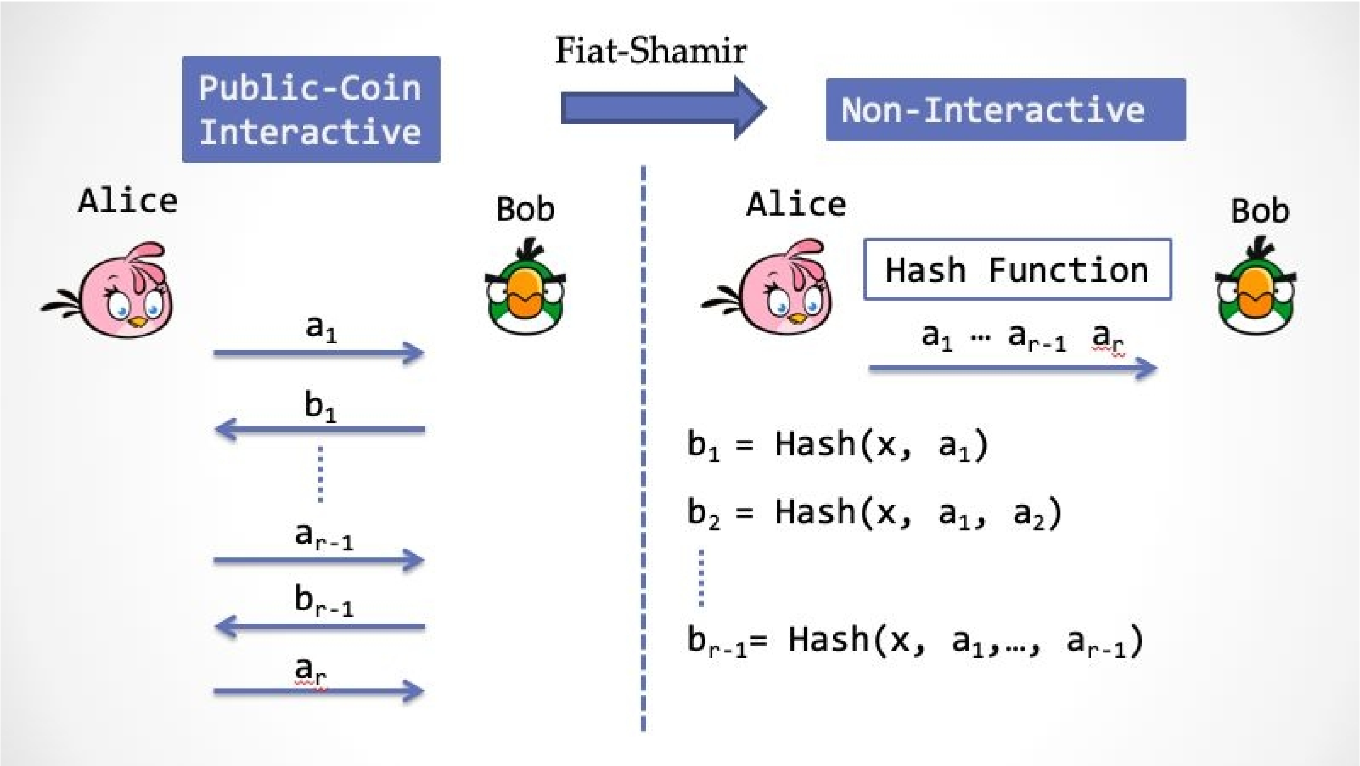
椭圆曲线加密随机数c的作用

如果Bob不回复一个c，就变成如下一次性交互。
一个攻击者完全可以在不知道a的情况下构造：R = r * G - PK 和 z = r。
这样Bob的验证过程就变成：z * G ?== PK + R ==| r * G ?== PK + r * G - PK。
这是永远成立的，所以这种方案并不正确。



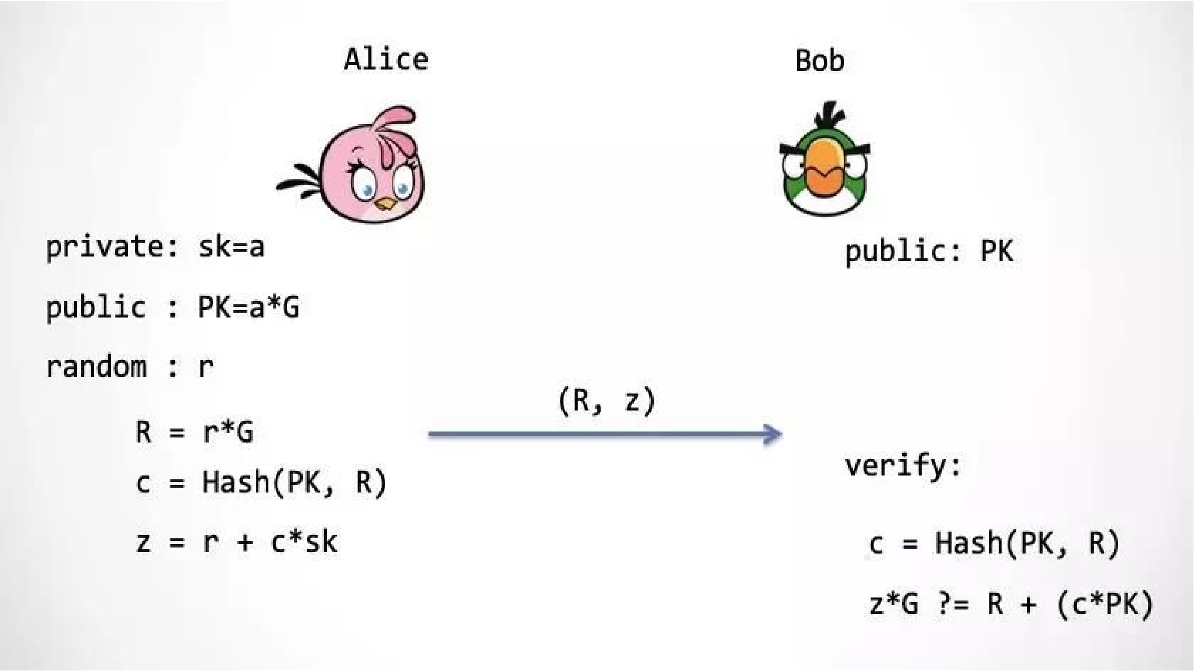
Fiat-Shamir变换

采用Hash函数的方法来把一个交互式的证明系统变成非交互式的方法被称为Fiat-Shamir变换，它由密码学老前辈Amos Fiat和Adi Shamir两人在1986年提出。
Fiat-Shamir变换，又叫Fiat-Shamir Heurisitc（启发式），或者Fiat-Shamir Paradigm（范式）。是Fiat和Shamir在1986年提出的一个变换，其特点是可以将交互式零知识证明转换为非交互式零知识证明。这样就通过减少通信步骤而提高了通信的效率！
菲亚特-沙米尔（Fiat-Shamir）启发式算法允许将交互步骤替换为非交互随机数预言机（Random oracle）。随机数预言机，即随机数函数，是一种针对任意输入得到的输出之间是项目独立切均匀分布的函数。理想的随机数预言机并不存在，在实现中，经常采用密码学哈希函数作为随机数预言机。



非交互schnorr

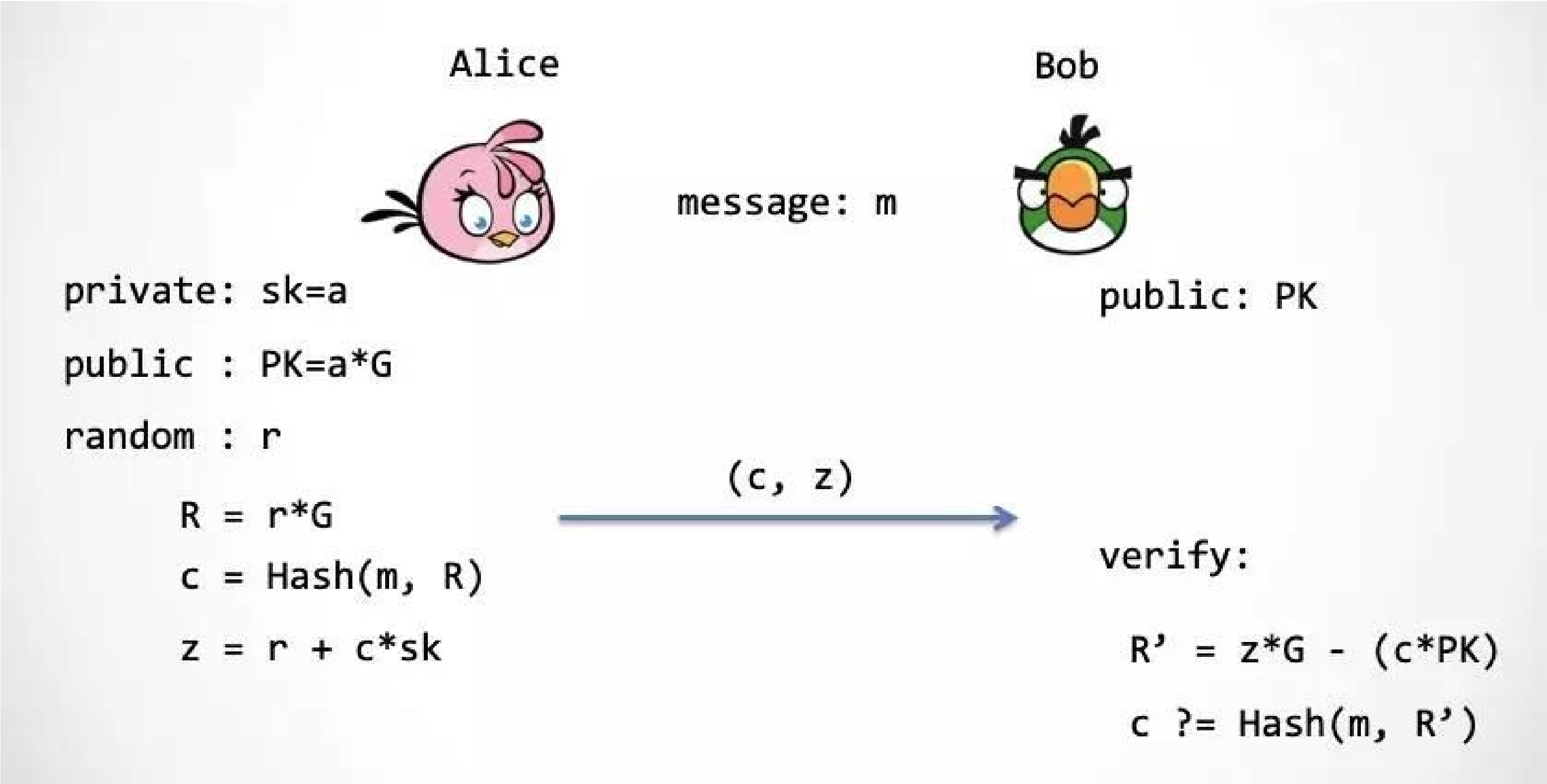
为了不让Alice进行造假，需要Bob发送一个c值，并将c值构造进公式中。所以，如果Alice选择一个无法造假并且大家公认的c值并将其构造进公式中，问题就解决了。生成这个公认无法造假的c的方法是使用哈希函数。
看一下交互式Schnorr协议的第二步，Bob需要给出一个随机的挑战数c，这里我们可以让Alice用c=Hash(PK,R)这个式子来计算这个挑战数，从而达到去除协议第二步的目的。
此外，利用Hash算法计c的式子，这里还达到了两个目的：
第一个目的：Alice在产生承诺R之前，没有办法预测c，即使c最终变相是Alice挑选的。
第二个目的：c通过Hash函数计算，会均匀分布在一个整数域内，而且可以作为一个随机数。
请注意：Alice绝不能在产生R之前预测到c，不然，Alice就等于变相具有了「时间倒流」的超能力，从而能任意愚弄Bob。



而一个密码学安全Hash函数是「单向」的，比如SHA256等。这样一来，虽然c是Alice计算的，但是Alice并没有能力实现通过挑选c来作弊。因为只要Alice一产生R，c就相当于固定下来了。我们假设Alice这个凡人在「现实世界」中是没有反向计算Hash的能力的。
这样，就把三步Schnorr协议合并为一步。Alice可直接发送（R，z），因为Bob拥有Alice的公钥PK，于是Bob可自行计算出c。然后验证z*G ?=c*PK+R。

Schnorr签名方案

在这里还有一个优化，Alice发给Bob的内容不是(R,z)而是(c,z)，这是因为R可以通过c,z计算出来。



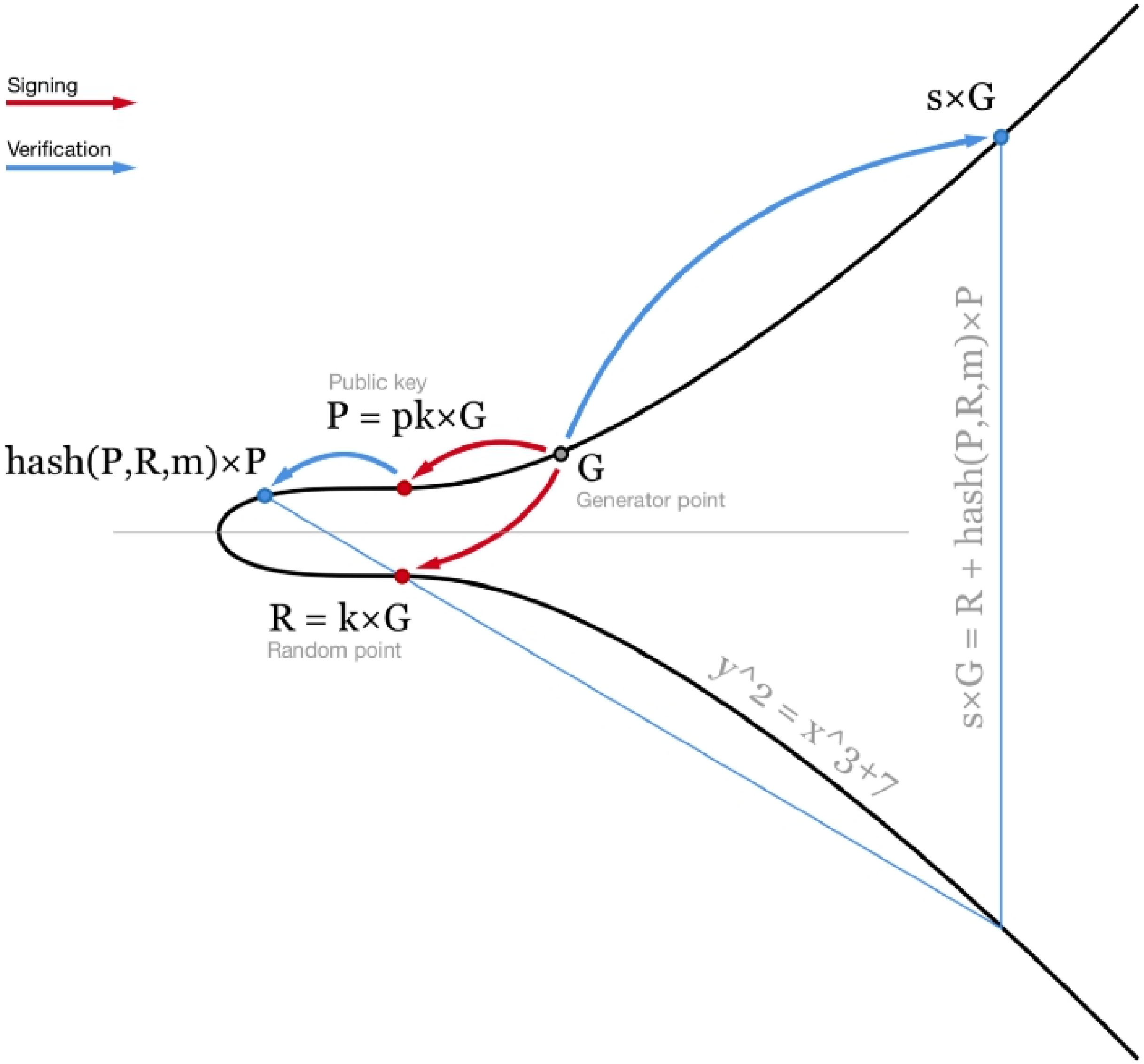
Schnorr签名图解

Schnorr签名使用点 R 和标量 s 来生成签名，R 是椭圆曲线上的一随机点：

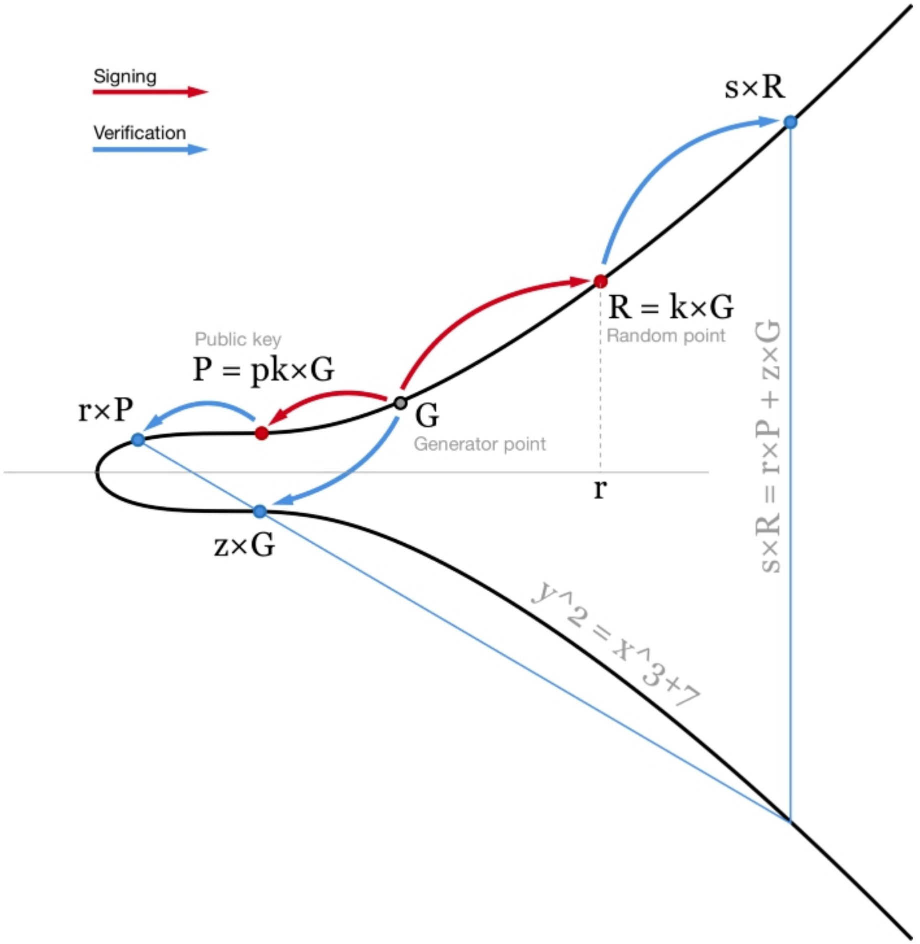
$R=k\cdot G$，签名第2部分为：$s=k+H(P,R,m)\cdot sk$，其中 sk 为私钥，

$P=sk\cdot G$ 为公钥，m 为消息，采用下式验证签名的有效性：

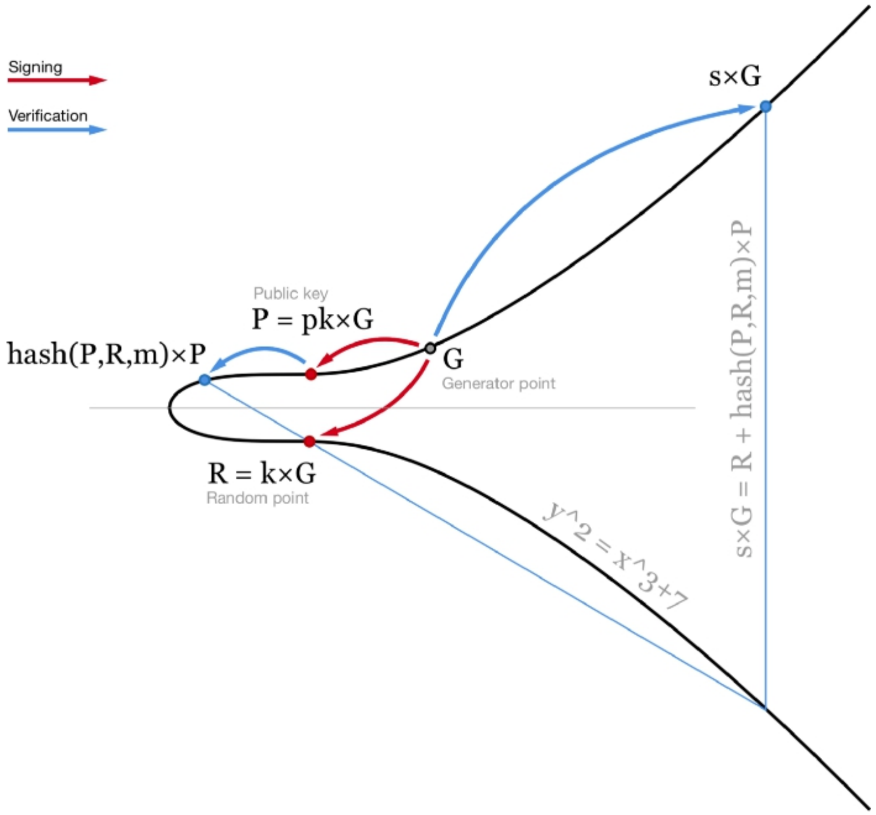
$$s\cdot G=R+H(P,R,m)\cdot P$$



Schnorr vs ECDSA

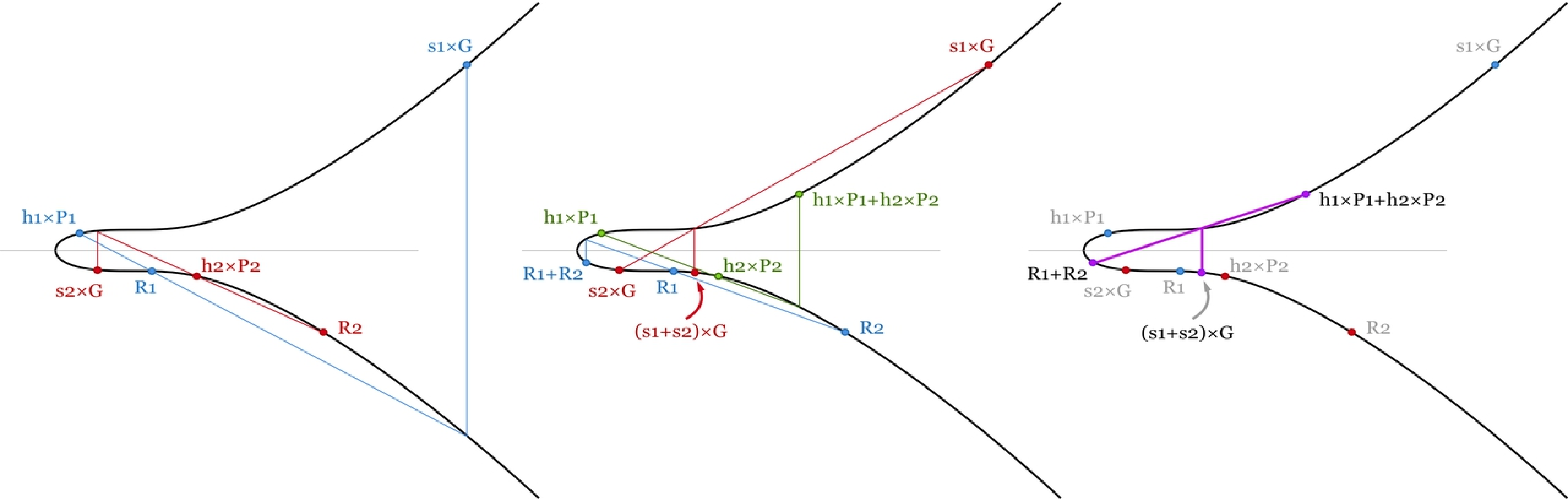


vs



Schnorr vs ECDSA

schnorr等式是线性的，所以多个等式可以相加相减而等号仍然成立。这给我们带来了 Schnorr 签名的多种良好特性。
在验证区块链上的一个区块时，我们需要验证区块中所有交易的签名都是有效的。如果其中一个是无效的，无论是哪一个 —— 我们都必须拒绝掉整个区块。
ECDSA 的每一个签名都必须专门验证，意味着如果一个区块中包含 1000 条签名，那我们就需要计算 1000 次除法和 2000 次点乘法，总计约 3000 次繁重的运算。
但有了 Schnorr 签名，我们可以把所有的签名验证等式加起来并节省一些计算量。在一个包含 1000 笔交易的区块中，我们可以验证：
$(s1+s2+\ldots+s1000)\times G=(R1+\ldots+R1000)+(hash(P1,R1,m1)\times P1+hash(P2,R2,m2)\times P2+\ldots+hash(P1000,R1000,m1000)\times P1000)$(s1+s2+…+s1000)×G=(R1+…+R1000)+(hash(P1,R1,m1)×P1+hash(P2,R2,m2)×P2+…+hash(P1000,R1000,m1000)×P1000)
这里就是一连串的点加法（从计算机运算的角度看，简直是免费的）和 1001 次点乘法。已经是几乎 3 倍的性能提升了 —— 验证时只需为每个签名付出一次重运算。



聚合签名的实现

有了 Schnorr 签名，我们可以使用一对密钥 (pk1,pk2)，并使用一个共享公钥 $P=P1+P2=pk1*G+pk2*G$P=P1+P2=pk1∗G+pk2∗G 生成一个共同签名。在生成签名时，我们需要在两个设备上分别生成一个随机数（k1, k2），并以此生成两个随机点 $Ri=ki*G$Ri=ki∗G，再分别加上 $hash(P,R1+R2,m)$hash(P,R1+R2,m)，就可以获得 s1 和 s2 了（因为 $si=ki+hash(P,R,m)*pki$ si=ki+hash(P,R,m)∗pki ）。最后，把它们都加起来即可获得签名 $(R,s)=(R1+R2,s1+s2)$(R,s)=(R1+R2,s1+s2)，这就是我们的共享签名，可用共享公钥来验证。其他人根本无法看出这是不是一个聚合签名，它跟一个普通的 Schnorr 签名看起来没有两样。

缺点-多轮交互：以热钱包+冷钱包双签为例
要发起一笔交易，我们需要在两个设备上发起多轮交互 —— 为了计算共同的 R，为了签名。在两把私钥的情况下，需要访问一次冷钱包：我们可以在热钱包里准备好待签名的交易，选好 k1 并生成 $R1=k1*G$R1=k1∗G，然后把待签名的交易和这些数据一同传入冷钱包并签名。因为已经有了 R1，签名交易在冷钱包中只需一轮就可以完成。从冷钱包中我们得到 R2 和 s2，传回给热钱包。热钱包使用前述的（k1，R1）签名交易，把两个签名加总起来即可向外广播交易了。

Rogue攻击

假设有两个参与者A和B，*PA*,*PB*分别是二者的公钥。
假设B不诚实，参与密钥聚合过程中，提供假的公钥$PFB=PB-PA$, 导致聚合公钥：
$P=PA+PFB=PA+PB-PA=PB$ ，
这样就控制了聚合公钥成为自己的公钥，从而只用B自己的签名来覆盖A的签名，本来需要A，B共同签名的交易，现在只要B单独签名（伪造聚合签名）就可以了。
这种攻击可称为“密钥消除攻击”，亦属于“Rogue Key Attacks”。

简单的解决方案是在密钥聚合操作中，参与者提供公钥所有权证明，即签署任意消息，但这会增加交互过程，如果这个所有权证明也放到区块链上，增加存储大小。

随机数攻击

攻击如下：某个黑客黑入你的笔记本电脑，完全控制了其中一把私钥（比如 pk1）。我们感觉资金仍是安全的，因为使用我们的比特币需要 pk1 和 pk2 的聚合签名。所以我们像往常一样发起交易，准备好一笔待签名的交易和 R1，发送给我们的硬件钱包，硬件钱包签名后将（R2, s2）发回给热钱包 …… 然后，热钱包出错了，没法完成签名和广播。于是我们再试一次，但这一次被黑的电脑用了另一个随机数 —— R1' 。我们在硬件钱包里签名了同一笔交易，又将（R2, s2'）发回给了被黑的电脑。这一次，没有下文了 —— 我们所有的比特币都不翼而飞了。
在这次攻击中，黑客获得了同一笔交易的两个有效的签名：（R1, s1, R2, s2）和（R1', s1'，R2，s2'）。这个 R2 是一样的，但是 $R=R1+R2$和 R'=R1′+R2 是不同的。这就意味着黑客可以计算出我们的第二个私钥：$s2-s2'=(hash(P,R1+R2,m)-hash(P,R1'+R2,m))\cdot pk2$

s2−s2′=(hash(P,R1+R2,m)−hash(P,R1′+R2,m))·pk2 或者说
$pk2=(s2-s2')/(hash(P,R1+R2,m)-hash(P,R1'+R2,$
$m))$pk2=(s2−s2′)/(hash(P,R1+R2,m)−hash(P,R1′+R2,m))。

这就是密钥聚合最不方便的地方 —— 我们每次都要使用一个好的随机数生成器，这样才能安全地聚合。

MuSig

聚合签名对应于聚合公钥，此时不将所有签名者公钥相加，而是乘以某个因子，聚合的公钥为：

$P = H(L, P_1) * P_1 + \cdots + H(L, P_n) * P_n$，此处

$L = H(P_1, \cdots, P_n)$ 为取决于所有公钥的公用数字，其非线性特性可以防止攻击者构造恶意的公钥。

为了生成签名，每个联合签名者选择一个随机数 k_i，并与其他人共享 $R_i = k_i * G$，再将这些随机点加在一起，得到 $R = R_1 + \cdots + R_n$， 生成签名 $s_i = k_i + H(P, R, m) \cdot H(L, P_i) \cdot sk_i$，生成的聚合签名为：$(R, s) = (R_1 + \cdots + R_n, s_1 + \cdots + s_n)$，验证方程为：

$$s \cdot G = R + H(P, R, m) \cdot P$$

总结一下，MuSig 签名分成三轮：

1. 所有参与者发送承诺 t_i
2. 所有参与者公开 nonce 数值 R_i，各方验证 $t_i = H(R_i)$
3. 所有参与者计算并发送自己的部分签名 s_i