M = message

x = priv key

G = group generator

P = pub key = $x \odot G$

k = random nonce

$R = k \odot G$ (Public nonce)

$s = k + Hash(R.P.M) \cdot x$

Signature is (R, s)

Cannot determine x from $P \& G$: $x = P/G$ but not defined discrete log problem

$\oplus$ = group op

$\odot$ = scalar $\odot$ Point $\rightarrow$ $P \oplus P \oplus \ldots \oplus P$ scalar times

EC property: x, y scalars w/ pts X, Y

$(x+y)G = xG + yG = X + Y$

CHECK

$sG = (k + Hash(R.P.M) \cdot x) \odot G = kG \oplus Hash(R.P.M)x \odot G =$

$R \oplus Hash(R.P.M) \odot P$ ← All available to the verifier, so compute this and compare to $s \odot G$, which we also have.

- **Why the nonce?**

No Nonce: $s = Hash(P.M) \cdot x$

Check $sG = Hash(P.M)x \odot G = Hash(P.M)P$

BUT: $x = \frac{s}{Hash(P.M)}$ VS. $x = \frac{s-k}{Hash(\ldots)}$

MODULAR SCALAR ARITHMETIC

Not feasible because k could be anything and cannot be derived from R if discrete log problem holds

- 2 signatures, same nonce? You can eliminate the nonce!

(R_0, s_0) $M_0 P_0$ | (R_1, s_1) $M_1 P_1$, $R_0 = R_1$ & $k_0 = k_1$

$s_0 - s_1 = k_0 + H_0 x - k_1 - H_1 x$

$s_0 - s_1 = H_0 x - H_1 x$

$s_0 - s_1 = (H_0 - H_1) x$

$\frac{s_0 - s_1}{H_0 - H_1} = x$ ← discovered private key!

- Why $Hash(R.P.M)$? Why not just m? $H(m)$ is more efficient because within 2^{256}.
- Why not $H(M)$? Answer: We can forge signatures to other messages. But this mapping

must have cryptographic hash properties or attacker could produce m_1 s.t. $H(m_1) = H(m)$ to make it seem like you signed a different message

$S = k + H(m) \cdot x$
$R = k \otimes G$

$S \otimes G = (k + mx) \otimes G$
$= k \otimes G + H(m)x \otimes G$
$= R + H(m) \otimes P$

$R = S \otimes G - H(m) \otimes P$

- Choose new Message m_1, pick a random S_1, calculate R_1

① $R_1 = S_1 \otimes G - H(m_1) \otimes P$

Propose (R_1, S_1) as a signature for m_1

Check (that choosing this R_1 fools the verifier)

Compute: $S_1 \otimes G$

Compute: $R_1 \oplus H(m_1) \otimes P =$

$(S_1 \otimes G - H(m_1) \otimes P) \oplus H(m_1) \otimes P$ by ①

$(S_1 \otimes G \oplus (H(m_1) - H(m_1)) \otimes P = S_1 \otimes G$ factor out $\otimes P$

$0 \rightarrow$ scalar subtraction, $0 \otimes P = I$, $X \oplus I = X$

If we hash just m_1, this expression (R_1) is computable. If we $H(m_1, R_1)$ this expression is NOT computable because we need to know R_1 in order to compute it.

★ We can forge signatures! (If we can calculate R_1 from publically known data)

We need to make S depend on R to ensure that we **cannot** take a random S and calculate R.

So: $S = H(R, M)$

- Why $H(R, P, M)$? Assume $H(\ldots) = H(R, M)$

$S \otimes G = R \oplus H(\ldots) \otimes P$

Answer: We can falsely

claim we signed a message

$S \otimes G - R = H(..) \otimes P$

$S \otimes G - k \otimes G = H(G) \otimes P$

$(S-k) \otimes G = H(R.m) \otimes P$

$\frac{(S-k)}{H(R.m)} \otimes G = P$

Choose a random k and calc x and P to claim a signature as your own.

Remember $x \otimes G = P$, so

$x = \frac{S-k}{H(R.m)}$

How this could be used is less clear, but for example, you could falsely "prove" that you signed a statement that someone else made. Since the blockchain commits to the pubkey or pubkey hash in the prior tx the attack vector is limited in a blockchain context.

Signer

Verifier

$(k + H(R.m)x) \otimes G = R \oplus H(R.m) \otimes P$

$(k + H(R.m)x) \otimes G = R \oplus H(R.m) \otimes P$