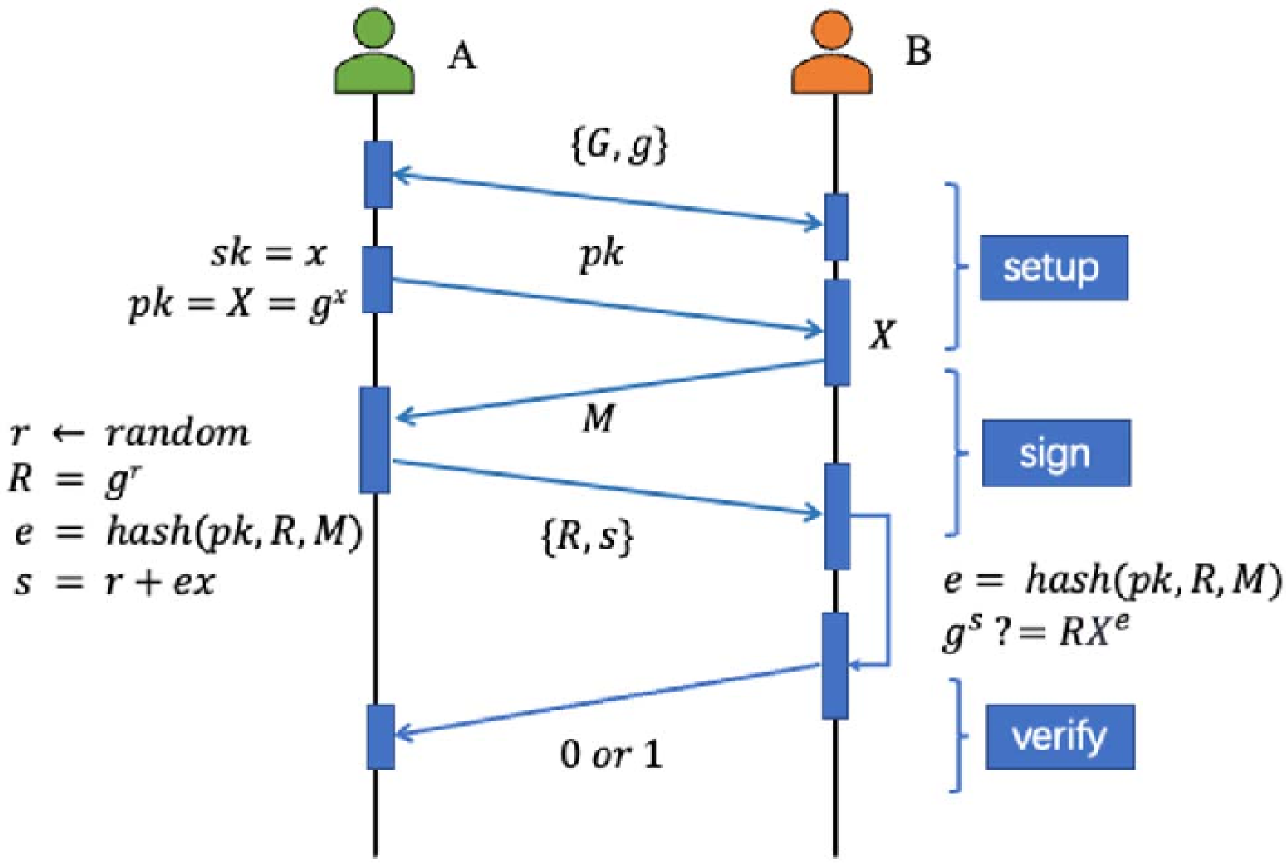
Schnorr's signature

- Schnorr signatures are a cryptographic signature scheme widely used in secure distributed systems.
- They provide a means to prove the authenticity and integrity of a message using mathematical principles.



Key Components

Public Key and Private Key

- Schnorr signatures use key pairs:
  - **Public Key (`P`)**: A point on an elliptic curve.
  - **Private Key (`x`)**: A randomly selected integer.

Signature Variables

- Several variables and values are involved in the Schnorr signature process:
  - **Nonce (`k`)**: A random value selected for each signature. It must be kept secret.

- **Nonce Point (`R`)**: Computed as `R = k * G`, where `G` is the generator point on the elliptic curve.
- **Message (`m`)**: The data to be signed.
- **Challenge (`e`)**: Calculated as `e = H(R . P . m)`, where `H` is a secure cryptographic hash function.
- **Signature Scalar (`s`)**: Computed as `s = k + e * x`.

Signing Process

- Creating a Schnorr signature involves several steps:
  - **Nonce Generation**: Select a random nonce `k`.
  - **Nonce Point**: Compute the nonce point `R = k * G`, where `G` is a fixed generator point on the elliptic curve.
  - **Challenge Computation**: Calculate the challenge `e` as `e = H(R . P . m)`.

Verification Process

- To verify a Schnorr signature:
  - The verifier independently computes `e` using the same inputs.
  - Check if `s * G = R + e * P`.

Mathematical Details

Scalar Multiplication

- Scalar multiplication involves adding a point to itself multiple times.
- It is a computationally intensive operation in the signature generation.

Hash Function

- A secure cryptographic hash function is used to generate the challenge `e`.
- It ensures that `e` is of a fixed size and derived from the public key, nonce point, and message.

Example

Signer's Perspective

- Private key: `d = 42`
- Random nonce: `k = 17`
- Elliptic curve parameters: Chosen ECDSA curve

1. Compute `R = k * G`.
2. Calculate the challenge `e = H(R . P . m)`.
3. Calculate `s = k + e * x`.
   **Signature is `(R,s)`

Verifier's Perspective

- Receive `P`, `m`, and `(s, R)`.

1. Compute the challenge `e = H(R . P . m)`.
2. Check if `s * G = R + e * P`.
   - If the equation holds, the signature is valid.

Security Considerations

- The security of Schnorr signatures relies on randomness, secure elliptic curve choice, and the use of a secure hash function.
- Proper key management is essential to maintain security.

These detailed notes and illustrations should help you understand the Schnorr signature scheme comprehensively and serve as a reference for your studies and work in secure distributed systems.

Notes and rough

```
![[Pasted image 20231023125447.png]]
```

Intuitions and proofs

1. Why nonce?

The below proof explains why the nonce is needed and how hard it makes to break the signature and find `x` with nonce present compared to without nonce.

Why do we need nonce?

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

$\boxed{\text{Signature is } (R, s)}$ ← Signing

$\boxed{s \circ G = R + Hash(R.P.m) \circ P}$ ← verifying

⇒ if there is no k ⇒ no R

Signing will be $\boxed{s = Hash(P.m)\, x}$

Verifying will be $\boxed{s \circ G = Hash(P.m)\, P}$ ✓

→ not secure!

$s = Hash(P.m)\, x$

Public, Public, Public

scalar, scalar, scalar

So I can just

$\frac{s}{Hash(P.m)} = $ I will get x

If we have k & R

This problem becomes little hard

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

To get x

Public → $\frac{s - k}{Hash(R.P.m)} = x$ ← Still private

Public

∴ since 'k' can be any large number
It's not feasible to get x

Can we get 'k' from R?

No, $R = k \circ G$
→ discrete log problem makes this Infeasible too.

2. What if 2 signatures have same nonce?

If two signatures use same nonce, that's another problem as one can discover private key from this.

What if 2 Signatures have same nonce?

(R_0, S_0), M_0 P_0 & (R_1, S_1), M_1, P_1

$P_0 = P_1$
as x is same

$k_0 = k_1$
$\Rightarrow R_0 = R_1$

we can derive x now
by first

$$S_0 - S_1 = K_0 + \underbrace{H(R_0.P.m_0)}_{H_0}x - k_1 - \underbrace{H(R_1.P.m_1)x}_{H_1}$$

$$S_0 - S_1 = \cancel{K_0} + H_0 x - \cancel{K_1} - H_1 x$$

$$S_0 - S_1 = (H_0 - H_1)x$$

$$\boxed{\therefore x = \frac{S_0 - S_1}{H_0 - H_1}}$$

$\Rightarrow$ we should always keep 'k' (nonce) random not constant.

3. Why Hash (R.P.m)? why not just m?

`H(m)` has more cryptographic advantage and is efficient. As the output of the hash is fixed length (generally 2^256). But we should make such the hash mapping should have cryptographic hash properties otherwise an attacker can
find a message m1 that hashes to the same hash as m and make it seem like you signed a different message

4. Why not H(m)?

If we use just H(m) **we can forge signatures to other messages**

what if we just use H(m)

Signature will be

$$S = K + H(m).x$$

$$R = K \odot G$$

verification

$$S \odot G = K \odot G + H(m) \odot P$$

$$\boxed{S \odot G = R + H(m) \odot P}$$

This eq'n makes R computable

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

FORGING signature

take a new message m_1 & a Random $S \Rightarrow S_1$

we can calculate R_1

with $= R_1 = S_1 \odot G - H(m_1) \cdot P$

& $\boxed{\text{propose } (R_1, S_1) \text{ as signature for } m_1}$

while verifying

$$S_1 \odot G = R_1 \oplus H(m_1) \odot P$$

$$= (S_1 \odot G - \cancel{H(m_1) \odot P}) + \cancel{H(m_1) \odot P}$$

↑ arithmetic

$$S_1 \odot G = S_1 \odot G$$

⇒ shows that S_1, R_1 become a valid signature for Pubkey P so, anyone can forge signatures for messages not by P, as P.

> We need to make S dependent on R thats why `H(R . m)` works as one cannot take a random `S1` and calculate `R1` since s and r are dependent. and R1 is part of hash's pre-image and that cant be solved to get R1

5. Why not H(R.m)? why only H(R . P . m)

- Why $H(R.P.M)$? Assume $H(\cdot) = H(R.m)$

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

Answer: We can falsely

[redacted]

claim we signed a message

$$S \circledast G - R = H(\cdot) \circledast P$$

$$S \circledast G - k \circledast G = H(\cdot) \circledast P$$

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

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

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

Remember $X \circledast G = P$, 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

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

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