Assinaturas no Bitcoin

Anfitrião:
Rafael Penna

bitcoinCoders bitups

O Que Veremos!

Assinaturas no Bitcoin

- Transações Bitcoin
  - Provar que é dono do UTXO
    - Chave privada associada a ele
    - Prova → assinatura digital
    - Gerada com a chave privada
    - Verificada publicamente com a chave pública

Assinaturas no Bitcoin

- Onde a assinatura aparece na transação
  - scriptSig

    scriptSig: <assinatura> <chave_publica>

    scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash>

    OP_EQUALVERIFY OP_CHECKSIG

  - Cíclo de validação

    UTXO anterior (output bloqueado)

    ↓ scriptPubKey

    Nova transação (tentando gastar)

    ↑ scriptSig

Assinaturas no Bitcoin

- O que é assinado exatamente?
  - Não assina toda a transação (apenas partes)
  - Tipo de assinatura (SIGHASH).
  - Byte extra → quais partes foram incluídas no hash antes da assinatura

Tipo	O que é assinado	Uso comum
`SIGHASH_ALL`	Todos os inputs e outputs	padrão
`SIGHASH_NONE`	Todos os inputs, nenhum output	transações dinâmicas
`SIGHASH_SINGLE`	Input i e output i	transações parciais
`ANYONECANPAY`	Apenas o input atual	gastos cooperativos

Assinaturas no Bitcoin

- Na prática
  - Transação na Signet (P2PKH)
  - Decodificando uma transação (Hex):

```
bitcoin-cli -datadir="." -rpcwallet="demo-p2pkh-legacy" decoderawtransaction
0200000001fcf2c8867d5b8b25846ac2069cc988155109c11c359ff565fac7daa1029
e76cd000000006a47304402205900c10f73c239b7d731ec704dc11937bff3bcbb560
94214c80c5e6d73fe8f70022030756bad8a00a9e1f2b971d6851eabf3cf0689c27b5
5d3efb8b1941b0d59dafb012102d0592672fe7b5d840f54eb47775eed9410433943
3377fbb1bc5a2818290b4f89fdffffff0158020000000000001976a914603c773f8f141
51e7e75e52d56074cf254359f7188ac00000000
```

Assinaturas no Bitcoin

- Na prática
  - Transação na Signet (P2PKH)
  - Decodificando uma transação (Hex):

"vin": […
"scriptSig": { "asm":
"304402205900c10f73c239b7d731ec704dc11937bff3bcbb56094214c80c5e6d73fe8f70022030756bad8a00a9e1f2b971d6851eabf3cf0689c27b55d3efb8b1941b0d59dafb[ALL]
02d0592672fe7b5d840f54eb47775eed9410433943337 7fbb1bc5a2818290b4f89",
…

Assinaturas no Bitcoin

- Na prática
  - Transação na Signet (P2PKH)
  - Decodificando uma transação (Hex):

```
"vout": [ { "value": 0.00000600, "n": 0, "scriptPubKey":
{ "asm": "OP_DUP OP_HASH160 603c773f8f14151e7e75e52d56074cf254359f71
OP_EQUALVERIFY OP_CHECKSIG", "desc":
... } }
```

Assinaturas no Bitcoin

- Assinando manualmente
  1.Criação da transação bruta

```
bitcoin-cli -datadir="." -rpcwallet="demo-p2pkh-legacy" \
  createrawtransaction
'[{"txid":"cd769e02a1dac7fa65f59f351cc109511588c99c06c26a84258b5b7d86c8f2
fc","vout":0}]' \
  '{"mpHobi2NgwhFpU7qyCgkTJFh9SZo63YFog":0.00000600}'
0200000001fcf2c8867d5b8b25846ac2069cc988155109c11c359ff565fac7daa1029
e76cd0000000000fdffffff01580200000000000001976a914603c773f8f14151e7e75e
52d56074cf254359f7188ac00000000
```

Assinaturas no Bitcoin

- Assinando manualmente
  2. Adição de taxa e troco

```
bitcoin-cli -datadir="." -rpcwallet="demo-p2pkh-legacy" \
  fundrawtransaction
"0200000001fcf2c8867d5b8b25846ac2069cc988155109c11c359ff565fac7daa102
9e76cd0000000000fdffffff0158020000000000001976a914603c773f8f14151e7e75
e52d56074cf254359f7188ac00000000" '{"feeRate":0.00000400}'

{  "hex":
"0200000001fcf2c8867d5b8b25846ac2069cc988155109c11c359ff565fac7daa102
9e76cd0000000000fdffffff0158020000000000001976a914603c773f8f14151e7e75
e52d56074cf254359f7188ac00000000",
  "fee": 0.00000400,  "changepos": -1
}
```

Assinaturas no Bitcoin

- Assinando manualmente
  2. Adição de taxa e troco
     Decodificando…

```
bitcoin-cli -datadir="." -rpcwallet=demo-p2pkh-legacy decoderawtransaction
0200000001fcf2c8867d5b8b25846ac2069cc988155109c11c359ff565fac7daa1029
e76cd0000000000fdffffff0158020000000000001976a914603c773f8f14151e7e75e
52d56074cf254359f7188ac00000000
{…
  "scriptSig": {
   "asm": "",
   "hex": ""
  },…
```

bitcoinCoders bitups

Assinaturas no Bitcoin

- Assinando manualmente
  3. Assinatura com a carteira

```
bitcoin-cli -datadir="." -rpcwallet="demo-p2pkh-legacy"
signrawtransactionwithwallet
0200000001fcf2c8867d…3f8f14151e7e75e52d56074cf254359f7188ac00000000

{
  "hex":
"0200000001fcf2c8867d5b8b…1976a914603c773f8f14151e7e75e52d56074cf254
359f7188ac00000000",
  "complete": true
}
```

Assinaturas no Bitcoin

- Assinando manualmente
  3. Assinatura com a carteira (decodificando)

```
bitcoin-cli -datadir="." -rpcwallet=demo-p2pkh-legacy decoderawtransaction
0200000001fcf2…51e7e75e52d56074cf254359f7188ac00000000

{ … "scriptSig": { "asm":
"304402205900c10f73c239b7d731ec704dc11937bff3bcbb56094214c80c5e6d73fe
8f70022030756bad8a00a9e1f2b971d6851eabf3cf0689c27b55d3efb8b1941b0d59
dafb[ALL]
02d0592672fe7b5d840f54eb47775eed9410433943337 7fbb1bc5a2818290b4f89",
… ],
```

Assinaturas no Bitcoin

- Assinando manualmente
  4. Transmitindo a transação

```
bitcoin-cli -datadir="." -rpcwallet="demo-p2pkh-legacy" sendrawtransaction
0200000001fcf2c8867d5b8…51e7e75e52d56074cf254359f7188ac00000000

3523ba016a13ba621223ffab879b1042494a0e707c8c61ba2285d6094276c96a
```

O Que Veremos!

ECDSA

- Elliptic Curve Digital Signature Algorithm
- Algoritmo de Assinatura Digital de Curvas Elípticas

- Assinatura é um selo matemático que liga três elementos:
  - A mensagem (no caso do Bitcoin, partes da transação)
  - A chave privada (segredo do proprietário)
  - O algoritmo de assinatura (ECDSA, no caso do protocolo)

- Resultado:
  - Par de números, normalmente chamados de (r, s)
  - Permitem a qualquer um verificar a autenticidade
  - Impossibilitam reconstruir a chave privada original

ECDSA

- Hashing ≠ Assinatura
  - Papéis distintos.
  - Hash
    - Operação unidirecional
    - “Impressão digital” única de uma mensagem
  - Assinatura digital
    - Operação assimétrica
      - Só chave privada pode gerar a assinatura
      - Qualquer um com a chave pública pode verificar

ECDSA

- Hashing ≠ Assinatura
  - Bitcoin
    - Hash da transação + assinatura
    - Depende do conteúdo exato da transação
    - Altera byte → invalida a assinatura

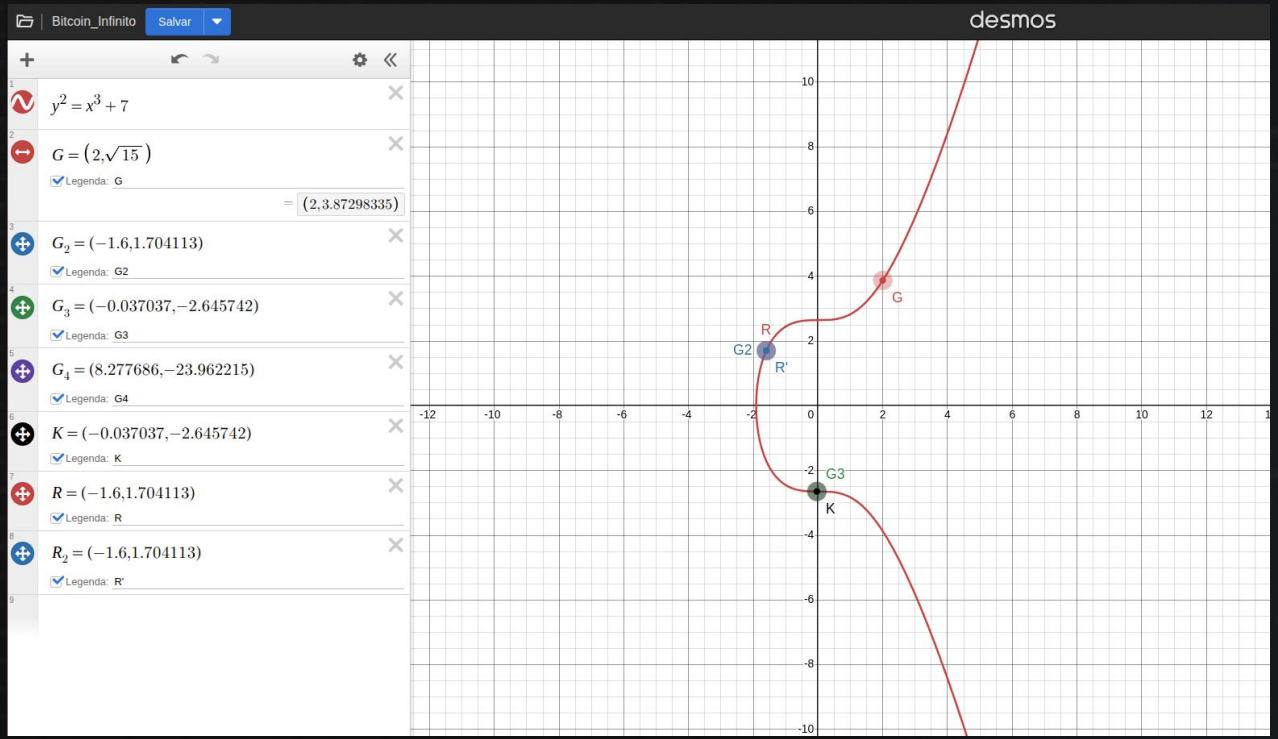
ECDSA

- A curva elíptica secp256k1
  - $y^2=x^3+ax+b$
  - a e b são constantes que determinam a forma da curva
  - Propriedades geométricas especiais
    - Adição de pontos
    - Aritmética usada na criptografia moderna
  - No Bitcoin:
    - $y^2=x^3+7$ → em um espaço finito e discreto
    - a = 0 e b = 7

ECDSA

- A curva elíptica secp256k1
  - secp → Standards for Efficient Cryptography Prime curve
  - 256 → tamanho das chaves privadas, em bits
  - k1 → indica uma família de curvas com parâmetros simples (koblitz curves)
  - campo finito → módulo um número primo enorme
  - Chave privada (d) → número aleatório entre 1 e n−1
  - Chave pública (Q) → multiplicação de d pelo ponto gerador G
    - Q=d×G

ECDSA

- A curva elíptica secp256k1 (campo infinito)
  - 

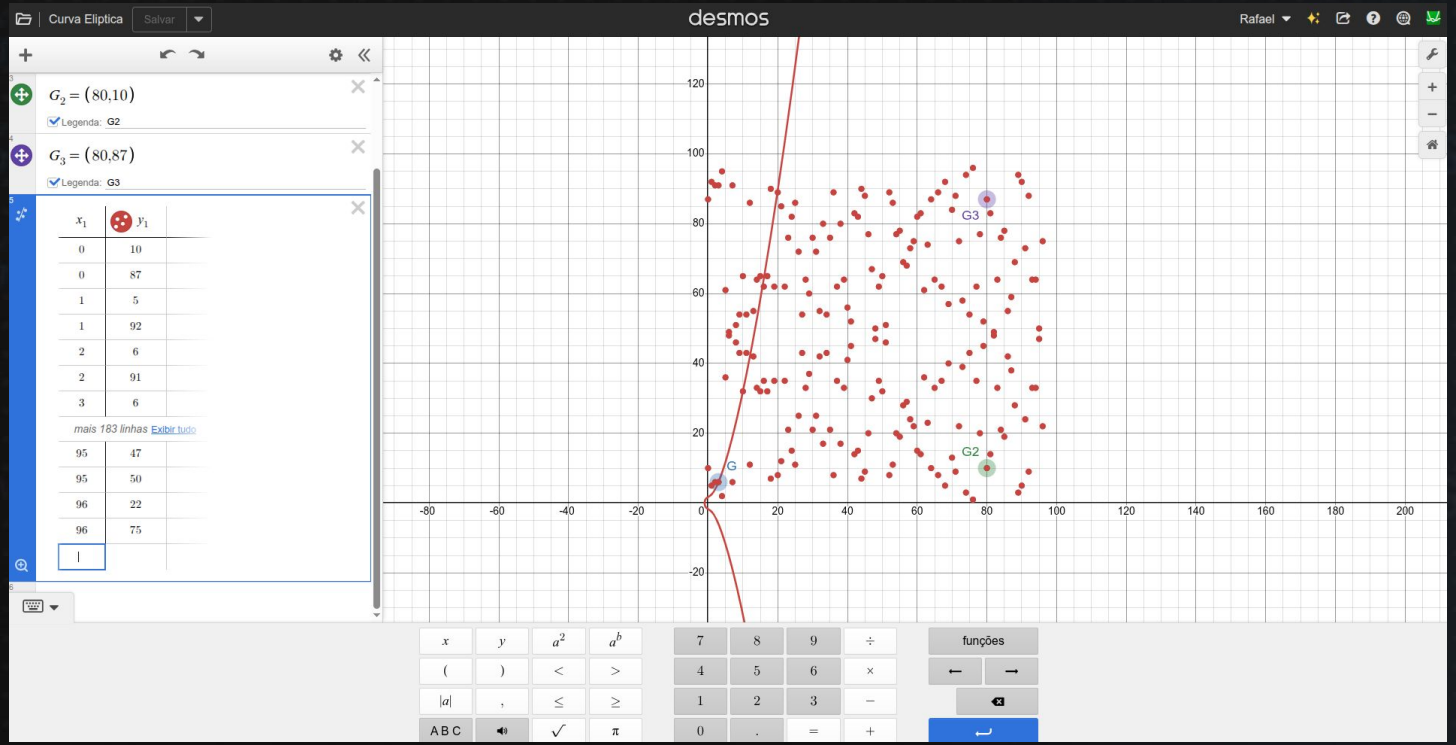
bitcoinCoders bitups

ECDSA

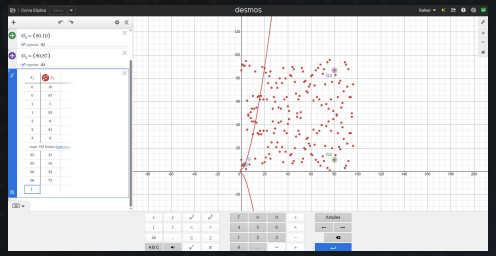
- A curva elíptica secp256k1 (campo infinito)
  - Curva (para visual): $y^2=x^3+7$
  - Ponto gerador: $G=(2, \sqrt{15})\approx(2.0, 3{,}872983)$
  - Múltiplos:
    - $2G\approx(-1.6, 1.704113)$
    - $3G\approx(-0.037037, -2.645742)$
    - $4G\approx(8.277686, -23.962215)$
  - Chave privada: $k=3$
  - Chave pública: $K=3G\approx(-0.037037, -2.645742)$
  - Nonce: $r=2$
  - Assinatura:
    - $r_x \rightarrow R=rG=2G\approx(-1.6, 1.704113)$
    - s (usado para reconstruir R’ na verificação)
  - Visualmente: $R_x'=R_x$ (verificador chega no mesmo x do ponto)
    - R’ = (-1.6, -1.70411267)

bitcoinCoders bitups

ECDSA

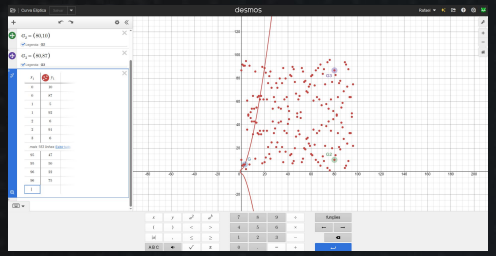
- A curva elíptica secp256k1 (campo finito)
  - 

ECDSA



- Variáveis para Assinatura (simulada/pequena):
  - Primo: p = 97
  - Curva: $y^2 \equiv x^3 + 2x + 3 \pmod{97}$
  - Ponto gerador: G = (3, 6)
  - Ordem do subgrupo gerado por G: n = 5
    - G = (3, 6)
    - 2G = (80, 10)
    - 3G = (80, 87)
    - 4G = (3, 91)
    - 5G = O (ponto no infinito, identidade)

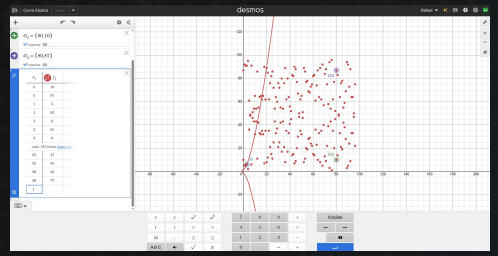
ECDSA



- Cálculo da Chave Pública (simulada/pequena):
  - Passo 1: escolher a chave privada
    - d = 2
    - Bitcoin → 256 bits
  - Passo 2: chave pública
    - K=$d \cdot G$
    - K=2G=(80,10)
  - 👉 Chave privada: d = 2
  - 👉 Chave pública: K = (80, 10)
  - secp256k1 faz exatamente isso, só que com números gigantes

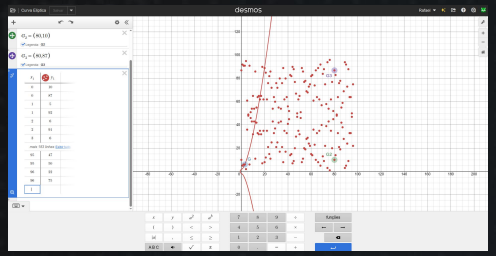
bitcoinCoders bitups

ECDSA



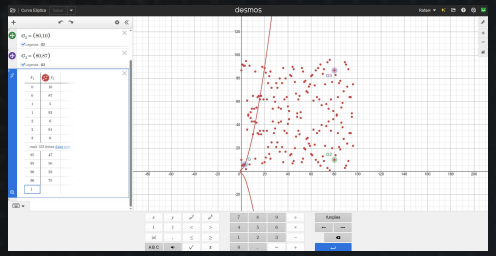
- Assinar uma mensagem (simulada/pequena):
  - “mensagem/transação” cujo hash (didático) é:
    - z = 2
  - Passo 1: escolher o nonce k
    - k = 1 (no mundo real precisa ser aleatório e secreto!)
  - Passo 2: calcular R = k·G
    - $R=kG=1G=(3,6)$
  - Passo 3: calcular r
    - $R_x=3 \rightarrow r=3$

ECDSA



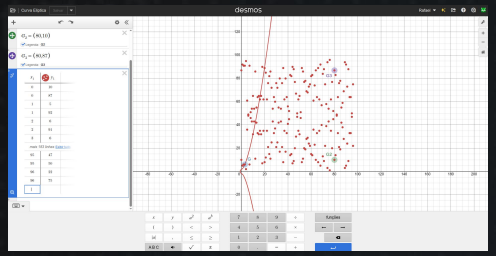
- Assinar uma mensagem (simulada/pequena):
  - Passo 4: calcular s
    - Fórmula ECDSA:
      - $s = k^{-1}(z + r \cdot d) \pmod{n}$
    - k = 1, z = 2, r = 3, d = 2
    - Calculando s:
      - $s = 1 \cdot (2 + 3 \cdot 2) \bmod 5$
      - $s = 8 \bmod 5$
      - $s = 3$
    - ✅ Assinatura = (r, s) = (3, 3)
  - No Bitcoin, é isso que vai parar no campo scriptSig / witness da transação (no formato DER).

ECDSA



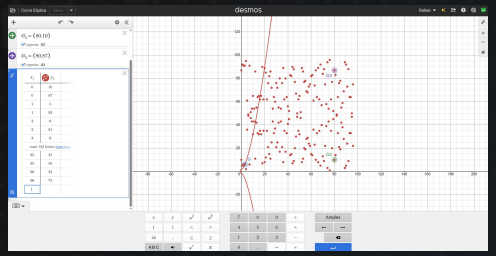
- Verificar a assinatura (simulada/pequena):
  - Alguém conhece só:
    - curva
    - p
    - gerador G
    - chave pública Q = (80, 10)
    - hash da mensagem z = 2
    - assinatura (r, s) = (3, 3)
  - Passo 1: calcular $w = s^{-1} \bmod n$
    - $s \cdot w \equiv 1 \pmod n$
    - $3 \cdot w \equiv 1 \pmod 5$
    - $w = 2$

ECDSA



- Verificar a assinatura (simulada/pequena):
  - Passo 2: calcular u_1 e u_2
    - $u_1 = z \cdot w \bmod n$
    - $u_2 = r \cdot w \bmod n$
    - $u_1 = 2 \cdot 2 \bmod 5 = 4$
    - $u_2 = 3 \cdot 2 \bmod 5 = 6 \bmod 5 = 1$
    - $u_1 = 4$
    - $u_2 = 1$
  - Passo 3: calcular o ponto X na curva
    - $X = u_1G + u_2K$
    - Sabemos que:
      - 4G = (3, 91)
      - K = (80, 10)
    - Somando na curva (soma elíptica mod 97):
      - $X = (3,6)$

ECDSA



- Verificar a assinatura (simulada/pequena):
  - Passo 4: teste final
    - A assinatura é válida se:
      - X_X mod n = r
    - Temos:
      - X_X = 3
      - 3 mod 5 = 3
      - 3 = 3
    - Logo:
      - ✅ Assinatura VÁLIDA

ECDSA

- A curva elíptica secp256k1
  - No Bitcoin:
    - Equação:
      - $y^2=x^3+7 \pmod p$
    - Primo:
      - $p=2^{256}-2^{32}-977$
    - Ponto gerador:
      - Gx=55066263022277343669578718895168534326250603453777594175500187360389116729240
      - Gy=32670510020758816978083085130507043184471273380659243275938904335757337482424

O Que Veremos!

Formato DER e SIGHASH

- ECDSA
  - (r e s)
  - Não têm forma definida

- DER (Distinguished Encoding Rules)
  - Interoperabilidade
  - Validação binária rígida
  - Padronização no protocolo

Formato DER e SIGHASH

- Estrutura do DER

  30 <len_total>
  02 <len_r> <r_em_big_endian>
  02 <len_s> <s_em_big_endian>

  - 30 → indica que vem uma SEQUENCE
  - Cada 02 → indica um INTEGER
  - Bytes <len_*>` → quantos bytes vêm a seguir para aquele campo

Formato DER e SIGHASH

- Estrutura do DER

  30 44
  02 20 59 00 c1 0f 73 c2 39 b7 d7 31 ec 70 4d c1 19 37
  bf f3 bc bb 56 09 42 14 c8 0c 5e 6d 73 fe 8f 70
  02 20 30 75 6b ad 8a 00 a9 e1 f2 b9 71 d6 85 1e ab f3
  cf 06 89 c2 7b 55 d3 ef b8 b1 94 1b 0d 59 da fb

  - 30 → tipo SEQUENCE (estrutura composta em ASN.1).
  - 44 (decimal 68) → comprimento total, em bytes, da sequência que vem depois: 02 20 <32 bytes de r/s> (2*(1 + 1 + 32) = 68)
  - 02 → tipo INTEGER
  - 20 → 32 bytes.
  - 32 bytes seguintes → valor de r (big-endian)
  - 32 bytes seguintes → valor de s (big-endian)

Formato DER e SIGHASH

- SIGHASH
  - scriptSig
    - <push_len> 30 ... <DER> <sighash_byte>
      47
        30 44 02 20 ... 02 20 ...
        01
    - 47 → opcode de push (71 bytes)
    - 30 44 ... → DER com r e s
    - 01 → SIGHASH_ALL
  - 01 → SIGHASH_ALL
  - 02 → SIGHASH_NONE
  - 03 → SIGHASH_SINGLE

Formato DER e SIGHASH

- Exemplo de scriptSig
  - bitcoin-cli -datadir="." decoderawtransaction "02000000010fa9052f266861c4496b81d695dbda14693ff0c525206ad05536df48a3acf5750010000006a47304402207e9a79a13fe534bf3c0f319748801ed4c9c3e9c75545f68eaf84d899428abee80220076b60df6fa63dfe8b5411753ef093afa3c3ad6b0918559ae093561b660b61cc012103d6d18224c3648a5a3d74367df185738bf02c82a92b2e7f7c84eb71a595300dda8fdffffff01250300000000000001976a9144e073e0dc8a9b26ae890b503d9c600e914c059d888ac00000000"
    "scriptSig": { "asm": "304402207e9a79a13fe534bf3c0f319748801ed4c9c3e9c75545f68eaf84d899428abee80220076b60df6fa63dfe8b5411753ef093afa3c3ad6b0918559ae093561b660b61cc[ALL] 03d6d18224c3648a5a3d74367df185738bf02c82a92b2e7f7c84eb71a59530dda8",
    "hex": "47304402207e9a79a13fe534bf3c0f319748801ed4c9c3e9c75545f68eaf84d899428abee80220076b60df6fa63dfe8b5411753ef093afa3c3ad6b0918559ae093561b660b61cc012103d6d18224c3648a5a3d74367df185738bf02c82a92b2e7f7c84eb71a59530dda8"
    }

bitcoinCoders bitups

Formato DER e SIGHASH

- Exemplo de scriptSig
  - 47 30 44 02 20 7e 9a 79 a1 3f e5 34 bf 3c 0f 31 97 48 80 1e d4 c9 c3 e9 c7 55 45 f6 8e af 84 d8 99 42 8a be e8 02 20 07 6b 60 df 6f a6 3d fe 8b 54 11 75 3e f0 93 af a3 c3 ad 6b 09 18 55 9a e0 93 56 1b 66 0b 61 cc
    01
    21 03 d6 d1 82 24 c3 64 8a 5a 3d 74 36 7d f1 85 73 8b f0 2c 82 a9 2b 2e 7f 7c 84 eb 71 a5 95 30 dd a8
- 47 → push de 71 bytes (assinatura DER + SIGHASH)
- 30 44 ... cc → assinatura em DER (r e s)
- 01 → SIGHASH_ALL
- 21 → push de 33 bytes (tamanho da chave pública)
- 03 d6 d1 82 ... dd a8 → chave pública comprimida

- P2PKH
  <assinatura_der + sighash> <chave_publica_comprimida>

O Que Veremos!

✍️

bitcoinCoders bitups

Schnorr e Taproot

- Taproot
  - Novembro de 2021
  - Novo algoritmo de assinatura (Schnorr)
  - Simplicidade matemática,
  - Eficiência
  - Múltiplas assinaturas em uma única prova compacta

Schnorr e Taproot

- Por que precisava evoluir do ECDSA?
  - Multisigs tradicionais revelem informações desnecessárias
    - Quantas chaves fazem parte do arranjo
    - Quais chaves participaram da assinatura
    - A ordem das chaves
    - Próprio fato de que aquilo é uma multisig
  - Transações complexas (scripts condicionais, timelocks e multisigs)
    - “impressão digital”
  - - privacidade
  - + espaço no bloco

Schnorr e Taproot

- Schnorr e ECDSA usam o mesmo
  - Grupo de pontos (curva elíptica)
  - G
  - p
  - Aritmética de pontos
  - Chave privada (inteiro)
  - Chave pública (ponto)

- O que é diferente
  - Como o nonce é combinado com a chave
  - Como o desafio é criado
  - Como a verificação é feita

Schnorr e Taproot

- Chave pública em Schnorr (mesma fórmula):
  - $P = d \cdot G$
  - d = chave privada (um inteiro)
  - P = chave pública (um ponto na curva)
  - G = ponto gerador
- R no Schnorr
  - $R = k \cdot G$
  - k = um número aleatório secreto (nonce)
  - G = ponto gerador da curva
  - R = é um ponto aleatório na curva, criado pelo assinante

Schnorr e Taproot

- Por que R existe?
  - Porque Schnorr funciona como um jogo:
    - Você se compromete com algo aleatório (R)
    - Recebe um desafio (e)
    - Responde ao desafio usando a chave privada
    - R é esse compromisso inicial.

Schnorr e Taproot

- Desafio
  - e=H(R // P // msg)

  - H = uma função hash criptográfica (SHA256)
  - R = o ponto aleatório kG
  - P = a chave pública do assinante (dG)
  - msg = a mensagem que será assinada

  - H mistura R, P e msg e transforma em um número desafiador e.
  - “Se você realmente é dono de P, mostre uma resposta que combine R, sua chave pública e essa mensagem específica”.

Schnorr e Taproot

- Passos do Assinante no Schnorr
  - Escolhe o nonce k (secreto, aleatório)
  - Calcula o ponto $R=k \cdot G$
    - Um ponto na curva gerado multiplicando G por k
    - É o compromisso inicial da assinatura
    - Vai aparecer na assinatura final
    - É público, mas não revela k.
  - Calcula o desafio $e=H(R \,//\, P \,//\, msg)$
    - Não faz parte da assinatura
  - Calcula a resposta $s=k+e \cdot d \pmod n$
    - Número inteiro
    - Combina: k (nonce),e (desafio) e d (chave privada)
    - Parte da assinatura que mostra que você conhece d, mas sem revelá-lo.
  - Publica a assinatura final (R, s)

Schnorr e Taproot

- Passos do Verificador no Schnorr
  - Recebe a assinatura (R, s)
  - Recalcula o desafio $e=H(R \,//\, P \,//\, msg)$
  - Calcula o lado esquerdo da equação: $LHS=s \cdot G$
    - s (parte da assinatura) e G (ponto gerador)
    - Resultado é um ponto na curva
  - Calcula o lado direito da equação: $RHS=R+e \cdot P$
    - R é o ponto da assinatura e P é a chave pública do assinante
    - e é o desafio reconstruído
    - eP significa multiplicar a chave pública por e
    - depois soma-se pontos na curva
    - “só bate se o assinante conhecia a chave privada.”
  - Compara os dois caminhos
    - $sG=R+eP$
    - Se forem iguais, a assinatura é válida

Schnorr e Taproot

- MAST
  - Antes do Taproot
    - UTXO com timelock, multisig e scripts complexos
      - Script completo → revelado
      - Mesmo que apenas uma das condições fosse usada
    - Isso significava:
      - Mais bytes
      - Menos privacidade
      - Scripts gigantes expostos
      - Custos maiores de validação
      - Padrões fáceis de identificar (ruim para privacidade)
  - Taproot resolve
    - MAST (Merkelized Abstract Syntax Tree)
      - Verificação → árvore Merkle
      - cada condição fica em um ramo isolado

Schnorr e Taproot

- Como MAST funciona
  1. Cada condição (script A, script B, script C…) vira uma folha

  2. Cada folha é hasheada individualmente → H(A), H(B), H(C)

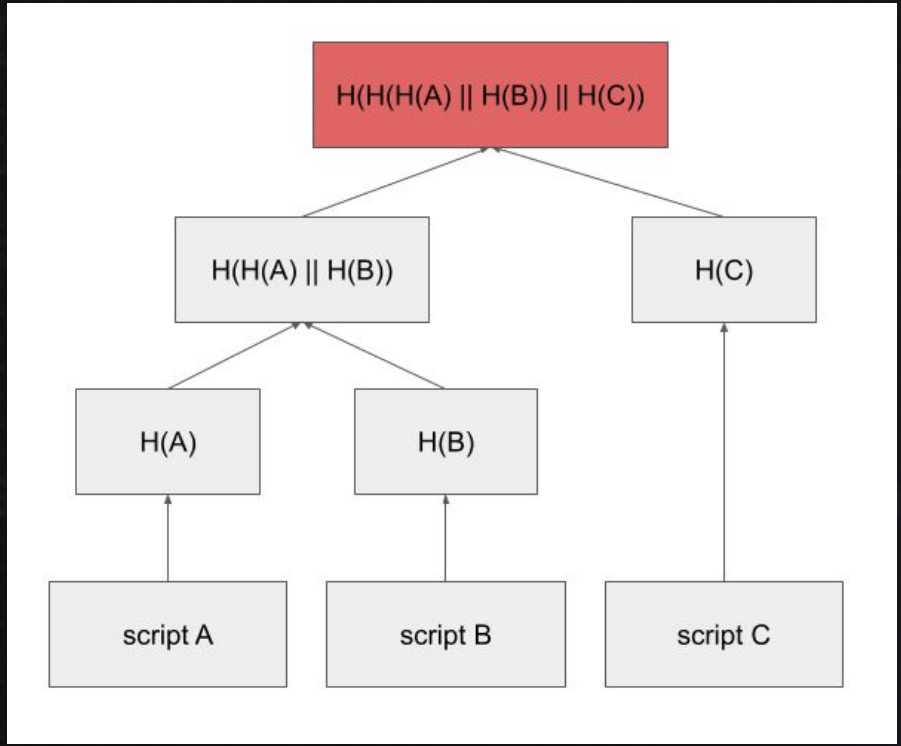
  3. As folhas são combinadas duas a duas (como um Merkle tree normal):
     - H(A) concatenado com H(B) → H(H(A) || H(B))

  4. Esse resultado é novamente combinado com o hash de H(C):
     - H(H(H(A)||H(B)) || H(C))

  5. O resultado final dessa árvore é o Merkle root, que será incorporado no tweak da chave Taproot.

Schnorr e Taproot

- Como MAST funciona



Schnorr e Taproot

- MAST é poderoso
  - Privacidade
    - Somente o ramo usado é revelado
  - Eficiência
    - Antes:
      - um contrato com 10 condições podia revelar 300–600 bytes de script.
    - Agora, apenas:
      - o script usado,
      - mais poucos hashes intermediários.
    - Em muitos casos, isso reduz dezenas ou centenas de bytes.
  - Modularidade
    - Contratos complexos → sem pagar nada extra se não forem usadas.

Schnorr e Taproot

- Como um gasto Taproot aparece no Bitcoin (Signet)

```
bitcoin-cli -datadir="." getrawtransaction
2d86980bc9b4f3b8b62dc771ef391647ab76e924265c22c387e2d09cddc393e
2 true
000000126586837 7fb0b0c0bb47cc1e6b5fb8d7db762b9b70640fe54f036f06f

"vin": [
  {
    "txinwitness": [

"81cebc1eb2db28511fb50de5b60d4888be5ff09294710793948 5e7ce1ae30ab6592
63ae39adb4fe7eb776b1d2e7ee60e5d307d7af2355632d997e30ca3579f41"
    ]
  }
]
```

Schnorr e Taproot

- Como um gasto Taproot aparece no Bitcoin (Signet)

"txinwitness": [
"81cebc1eb2db28511fb50de5b60d4888be5ff09294107939485e7ce1ae30ab659263ae39adb4fe7eb776b1d2e7ee60e5d307d7af2355632d997e30ca3579f41"]

- Há somente 1 elemento no witness
- Esse elemento é uma assinatura Schnorr
- A assinatura tem exatamente 64 bytes
- Não há chave pública
- Não há script
- Não há sighash explícito

- [32 bytes de R_x] [32 bytes de s]

FIM

Obrigado!

Grupo Whatsapp - Bitup Coders
https://chat.whatsapp.com/IB3rBamPe6QIvfncMBb3nF?mode=ems_copy_c

Rafael Penna
rapennas@gmail.com

bitcoinCoders bitups