section 04: public keys with python and bitcoin

public key math with python

but what is a public key?

simple, it's just a point on an elliptic curve, `secp256k1` in this case, over a finite field F_p.

remember:

`G * k -> P`, where `G` is the generator point, `k` is the private key (mind that `k < n` must hold) and `P` is the public key.

let's create a key pair in python, using the `coincurve` library, a python binding to `libsecp256k1`:

```
import coincurve

k = 777

privkey = coincurve.PrivateKey.from_int(k)

# compressed public key derived from the private key k=777
privkey.public_key.format().hex()
>'03e1fe434d345bf33083abb6280f4f44ac5fb22934977813c20c015f2b43d3fab8'

# (x, y) of the public key point on the EC
privkey.public_key.point()
>(102219634584123840528003342657511800276942438468621242403303479531589799770808,
 122780179549592841837198112019253953434855727142454693447580451977026179106 59)
```

so what `coincurve` does under the hood is multiply `G` by `k=777`, the private key we chose. this is shown in the code below:

```
# this is the G from libsecp256k1's spec
G = coincurve.PrivateKey.from_int(1).public_key

G.format().hex()
>'0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798'

# P = k*G
k = 777
P = G.multiply((k).to_bytes(32, 'big'))

P.format().hex()
>'03e1fe434d345bf33083abb6280f4f44ac5fb22934977813c20c015f2b43d3fab8'

P.point()
```

```
>(102219634584123840528003342657511800276942438468621242403303479531589799770808,
 122780179549592841837198112019253953434855727142454693447580451977026179106 59)
```

notice that both `privkey.public_key.point()` and `P.point()` yield the same result.

what public key cryptography proves

public key, or asymmetric, cryptography solves the problem of sharing a encryption key over a compromised communication channel.

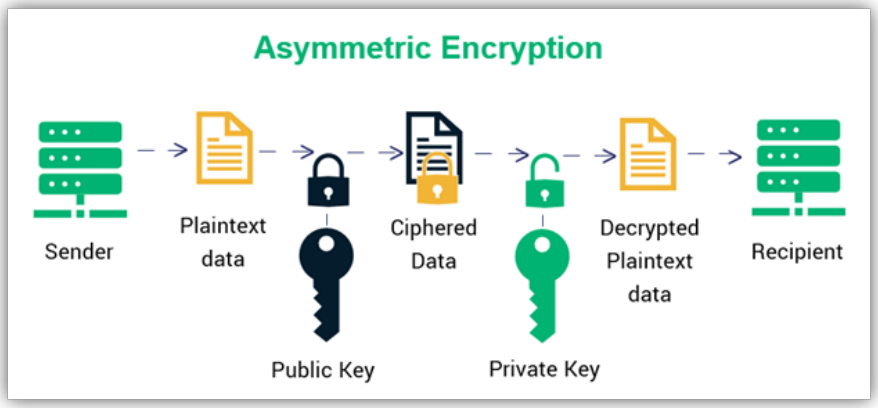
before, you needed to share a secret used to encrypt further messages with your counterpart, wich is very inconvenient, given you could be half way across the world from them. if this secret was leaked, either due to a MITM attack, or a rogue actor, your communication is compromised and you won't even know it.

with public key crypto, you generate a private key, and then derive a public key from it. you them freely share the public key with the world. whoever wants to send you a message, encrypts it with this key, and sends it to you. however, only your private key is capable of decrypting it.

this solves the **key exchange problem**. this process is called **Diffie-Hellman key exchange**.

moreover, your private key is capable of signing messages that can then be verified to be signed by you, using the public key.

eg: you share your pubkey with your friends. now, you want to sign a business agreement with your friend. you then sign it with your privkey. your friend can verify that it was indeed your privkey that signed it, but using only your pubkey.



Pay To Public Key [P2PK]

P2PK is a script used to lock up bitcoin to a specific public key. it works like this:

```
<pubkey>  OP_CHECKSIG
```

let k be a private key with value `k = 888`

the corresponding compressed pubkey is
`03c8edf725b59d9f370870a5dcd5e70b0baea493a5b85cb227c1ef6a8e8cdad8fb`

a compressed pubkey is `1` (parity) + `32` (data) = `33` bytes long, which is `0x21`.

the value for `OP_CHECKSIG` is `0xAC`.

so the script will look like this:

```
<lenght> <pubkey> <op code>

2103c8edf725b59d9f370870a5dcd5e70b0baea493a5b85cb227c1ef6a8e8cdad8fbac
```

if you run

```
bitcoin-cli -regtest -decodescript
2103c8edf725b59d9f370870a5dcd5e70b0baea493a5b85cb227c1ef6a8e8cdad8fbac
```

it yields:

```
{
  "asm": "03c8edf725b59d9f370870a5dcd5e70b0baea493a5b85cb227c
  1ef6a8e8cdad8fb OP_CHECKSIG",
  "desc": "pk(03c8edf725b59d9f370870a5dcd5e70b0baea493a5b85cb
  227c1ef6a8e8cdad8fb)#uyjfg78q",
  "type": "pubkey",
  "p2sh": "2N5qkyrMHFs2owkwkf76d55YmSMTX5YJa58",
  "segwit": {
    "asm": "0 2179dbd7258824b83b13e6190efdc302723183ec",
    "desc":
        "addr(bcrt1qy9uah4e93qjtswcnucvsalwrqferrqlvzrgycj)#mjpsey9y",
    "hex": "00142179dbd7258824b83b13e6190efdc302723183ec",
    "address": "bcrt1qy9uah4e93qjtswcnucvsalwrqferrqlvzrgycj",
    "type": "witness_v0_keyhash",
    "p2sh-segwit": "2N3h5LwNuYMcRzWziqSDTmB7o67QDDN1Z9W"
  }
}
```

let's make a transaction by hand to a pubkey (there is no address format for the P2PK script format):

```
version: 0002
input: 01
  txid:
      1c67a6d9f1638b1cce4359c3405114d53d749c2fe00581696336d6f8d21cc52b
  vout: 00000000
  scriptSig: 00
  sequence: fdffffff
output: 01
  amount: d4f0052a01000000
  scriptPubKey:
      232103c8edf725b59d9f370870a5dcd5e70b0baea493a5b85cb227c1ef6a8e8cdad8fbac
locktime: 00000000
```

now, we get rid of everything that's not a hex character:

```
0200011c67a6d9f1638b1cce4359c3405114d53d749c2fe00581696336d6f8d21cc52b000fdffffff
01d4f0052a01000000232103c8edf725b59d9f370870a5dcd5e70b0baea493a5b85cb227c1ef6a8e8
cdad8fbac00000000
```

now let's sign it:

```
bitcoin-cli -regtest
signrawtransactionwithwallet
0200011c67a6d9f1638b1cce4359c3405114d53d749c2fe00581696336d6f8d21cc52b0000000000f
dffffff01d4f0052a01000000232103c8edf725b59d9f370870a5dcd5e70b0baea493a5b85cb227c1
ef6a8e8cdad8fbac00000000`
```

this yields the following signed tx:

```
{
  "hex":
  "020000000001011c67a6d9f1638b1cce4359c3405114d53d749c2fe00581696336d6f8d21cc52b
  0000000000fdffffff01d4f0052a01000000232103c8edf725b59d9f370870a5dcd5e70b0baea49
  3a5b85cb227c1ef6a8e8cdad8fbac02473044022015e0a11ab4186d4b9ab98b69b4f5400098f5ab
  2eb7a61ba0caf2a152b3c0881c02201aba9772aa7311fbac743a9de1755860c0079fbd4bf413557
  684b206d93277f2012102672d70b977c8246cfa7c2df1fc6d9fbc3927ade3de90f51c602a4f32f6
  edb32c00000000",
  "complete": true
}
```

with this txid: `70a0e29dd3b67a934dec23938ca30db6955e0ef8428f2570f6227392dd8003b9`

unlocking from a P2PK script

ok, we know how to pay, or lock, bitcoin to a public key. but how do we spend, unlock, from it?

let's create a transaction to spend the UTXO created above, by hand:

to get a new address and scriptPubKey:

```
$ bitcoin-cli -regtest getnewaddress
bcrt1qmm7vljx7mth4f00umc2r6swchr52gdut5jtdhj

$ bitcoin-cli -regtest getaddressinfo
    bcrt1qmm7vljx7mth4f00umc2r6swchr52gdut5jtdhj
{
  "address": "bcrt1qmm7vljx7mth4f00umc2r6swchr52gdut5jtdhj",
  "scriptPubKey": "0014defccfc8dedaef54bdfcde143d41d8b8e8a4378b",
  "ismine": true,
  "solvable": true,
  "desc": "wpkh([9dbc6d1b/84'/1'/0'/0/2]
  0359d4f277ed9174de87c50f4155307e0ee78152c9fdd0a572dbd664c4f0a6448d)#4wa3f6gc",
  "parent_desc": "wpkh([9dbc6d1b/84'/1'/0']
  tpubDCZJWxNrUXopnZAkhNESExekvLm3a3beQ7cRCL4BPMjJtu6YpVGt2Az3GFodHihsuzx8eKSU4nn
  UQ131ZxeSDAqiapNA9R9QyJRcJaT558g/0/*)#mgv0pz46",
  "iswatchonly": false,
  "isscript": false,
  "iswitness": true,
  "witness_version": 0,
  "witness_program": "defccfc8dedaef54bdfcde143d41d8b8e8a4378b",
  "pubkey":
      "0359d4f277ed9174de87c50f4155307e0ee78152c9fdd0a572dbd664c4f0a6448d",
  "ischange": false,
  "timestamp": 1709583414,
  "hdkeypath": "m/84'/1'/0'/0/2",
  "hdseedid": "0000000000000000000000000000000000000000",
  "hdmasterfingerprint": "9dbc6d1b",
  "labels": [
    ""
  ]
}
```

now assemble the transaction, spending from the last txid (remember to convert it to little-endian):

```
version: 0200000
input: 01
  txid:
      b90380dd927322f670258f42f80e5e95b60da38c9323ec4d937ab6d39de2a070
  vout: 00000000
  scriptSig: 00
  sequence: fdffffff
output: 01
amount: a8ef052a01000000
scriptPubkey: 16 0014defccfc8dedaef54bdfcde143d41d8b8e8a4378b
```

```
locktime: 00000000
```

leaving only hexadecimals:

0200000001b90380dd927322f670258f42f80e5e95b60da38c9323ec4d937ab6d39de2a0700000000
000fdffffff01a8ef052a01000000160014defccfc8dedaef54bdfcde143d41d8b8e8a4378b00000000

now, let's sign this transaction with `signrawtransactionwithkey`. we must convert the private key `k=888` to base58,
which yields `cMahea7zqjxrtgAbB7LSGbcQUr1uX1ojuat9jZodMN8EvKBjKEAz`.

you can use base58.school/tools/wif to convert it.

```
bitcoin-cli -regtest signrawtransactionwithkey
0200000001b90380dd927322f670258f42f80e5e95b60da38c9323ec4d937ab6d39de2a0700000000
000fdffffff01a8ef052a01000000160014defccfc8dedaef54bdfcde143d41d8b8e8a4378b000000
00fcde143d41d8b8e8a4378b00000000
'["cMahea7zqjxrtgAbB7LSGbcQUr1uX1ojuat9jZodMN8EvKBjKEAz"]'
```

which yields:

```
0200000001b90380dd927322f670258f42f80e5e95b60da38c9323ec4d937ab6d39de2a0700000000
04847304402201ccd5a278e0805dc2634f3d6e713032a1921e1fc1245ee4153b166821a5bd39c0220
4879b07d7a5c5fc07fa3acae46b67bc58638ec586190066fb7f487f7e4bb734d01fdffffff01a8ef0
52a01000000160014defccfc8dedaef54bdfcde143d41d8b8e8a4378b00000000
```

then send it:

```
bitcoin-cli -regtest sendrawtransaction
0200000001b90380dd927322f670258f42f80e5e95b60da38c9323ec4d937ab6d39de2a0700000000
04847304402201ccd5a278e0805dc2634f3d6e713032a1921e1fc1245ee4153b166821a5bd39c0220
4879b07d7a5c5fc07fa3acae46b67bc58638ec586190066fb7f487f7e4bb734d01fdffffff01a8ef0
52a01000000160014defccfc8dedaef54bdfcde143d41d8b8e8a4378b00000000

>fc2c8a7a61bf231fc223af3fc5acfda3eff69252fac99b75d770369f5d9d49f6
```