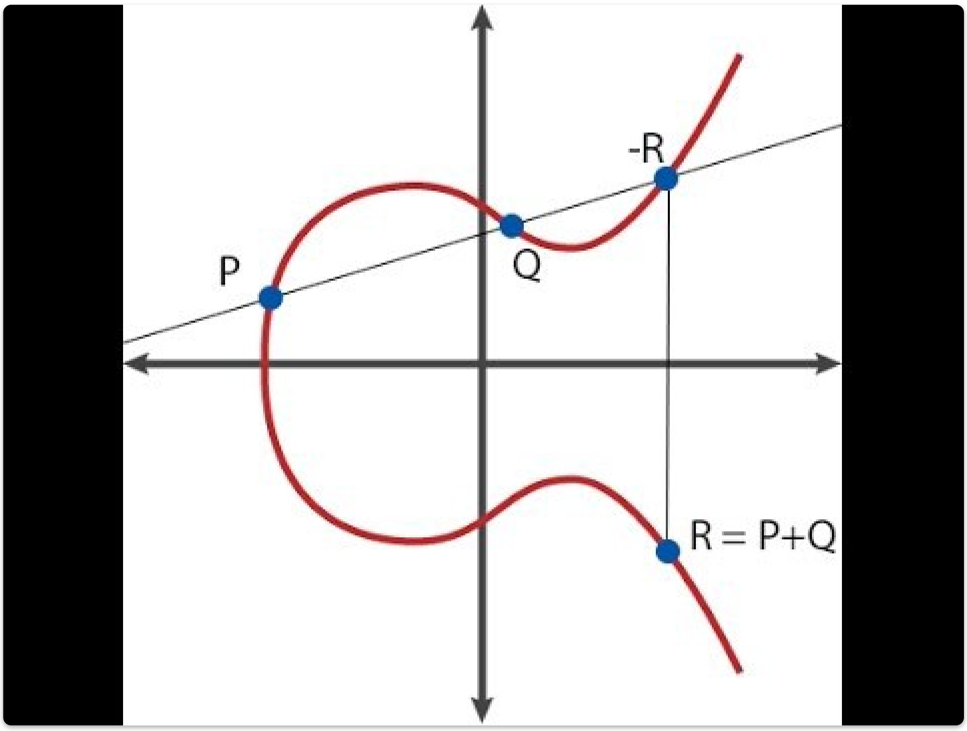
Q & A

Signature Malleability

ECDSA

The defining equation for an elliptic curve is for example $y^2 = x^3 + ax + b$



Ethereum uses the SECP-256k1 curve.

Cryptographic Signatures

See appendix F of the Ethereum Yellow Paper

Inputs

- Input message
- Private Key
- Random secret

Output

- Digital signature

The process can be checked even though the private key and random secret remain unknown.

Ethereum uses Elliptic Curve Digital Signature Algorithm, or ECDSA.
Note that ECDSA is only a signature algorithm. Unlike RSA and AES, it cannot be used for encryption.

Signing and verifying using ECDSA

ECDSA signatures consist of two integers: `r` and `s`.
Ethereum also uses an additional `v` (recovery identifier) variable. The signature can be notated as `{r, s, v}`.

To create a signature you need the message to sign and the private key (d_a) to sign it with. The "simplified" signing process looks something like this:

1. Calculate a hash (`e`) from the message to sign.
2. Generate a secure random value for `k`.
3. Calculate point (x_1, y_1) on the elliptic curve by multiplying `k` with the `G` constant of the elliptic curve.
4. Calculate $r = x_1 \bmod n$. If `r` equals zero, go back to step 2.
5. Calculate $s = k^{-1}(e + rd_a) \bmod n$. If `s` equals zero, go back to step 2.

In Ethereum in step 1 we usually add
`Keccak256("\x19Ethereum Signed Message:\n32")`
to the beginning of the hashed message.

To verify the message you need

- the original message
- the address associated with the private key
- the signature `{s,r,v}`

v is either 27 or 28 in Bitcoin and Ethereum before EIP 155, since then, the chain ID is used in the calculation of v, to give protection against replaying transactions
`v = {0,1} + CHAIN_ID * 2 + 35`

Why do we need the v value ?
There can be up to 4 different points for a particular x coordinate modulo n
2 because each X coordinate has two possible Y coordinates (reflection in x axis), and
2 because r+n may still be a valid X coordinate
The v value is used to determine which one of the 4.

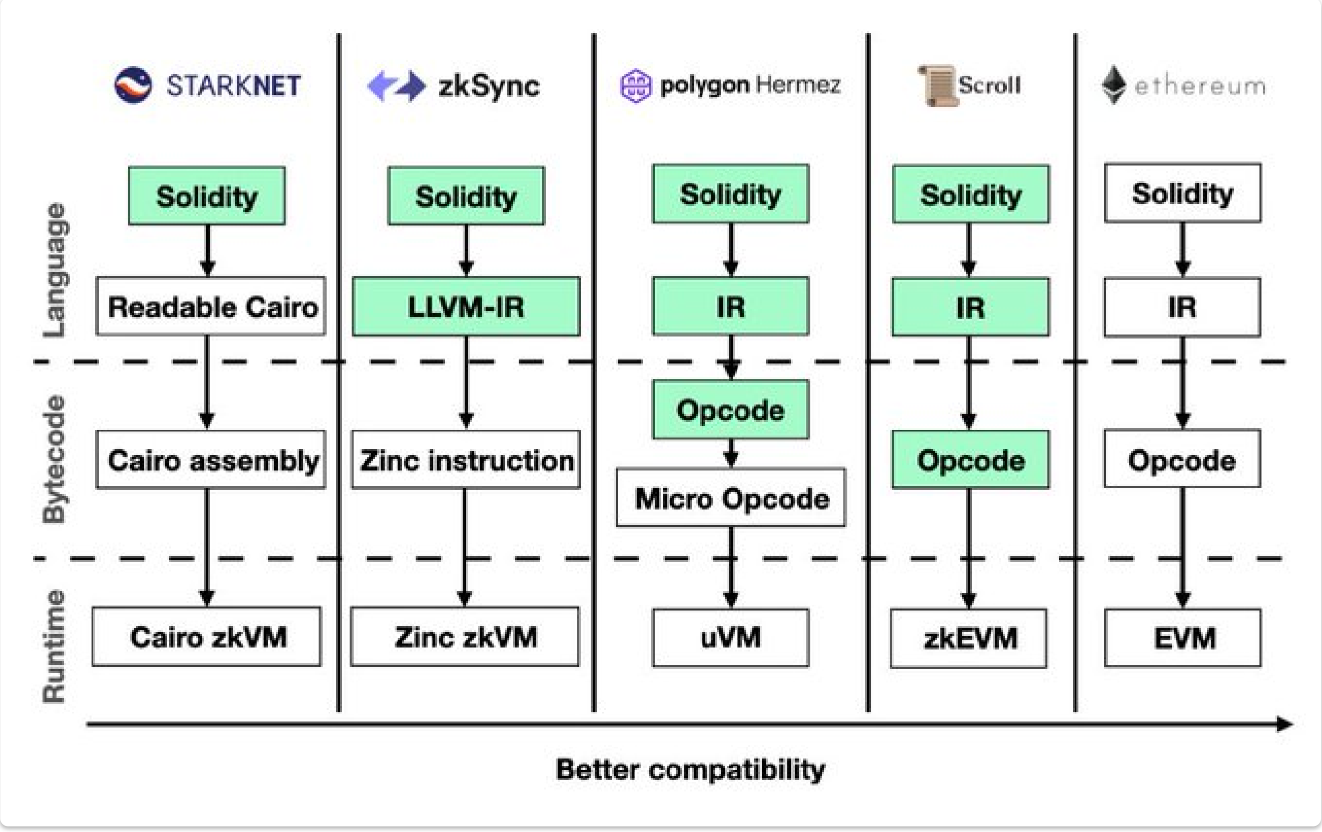
From the yellow paper
Recovery :
`ECDSARECOVER(e , v , r , s ) ≡ public key`
Where the public key is assumed to be a byte array of size 64
e is the hash of the transaction, h(T).
v, r, s are the values taken from the signature as above.

Malleability

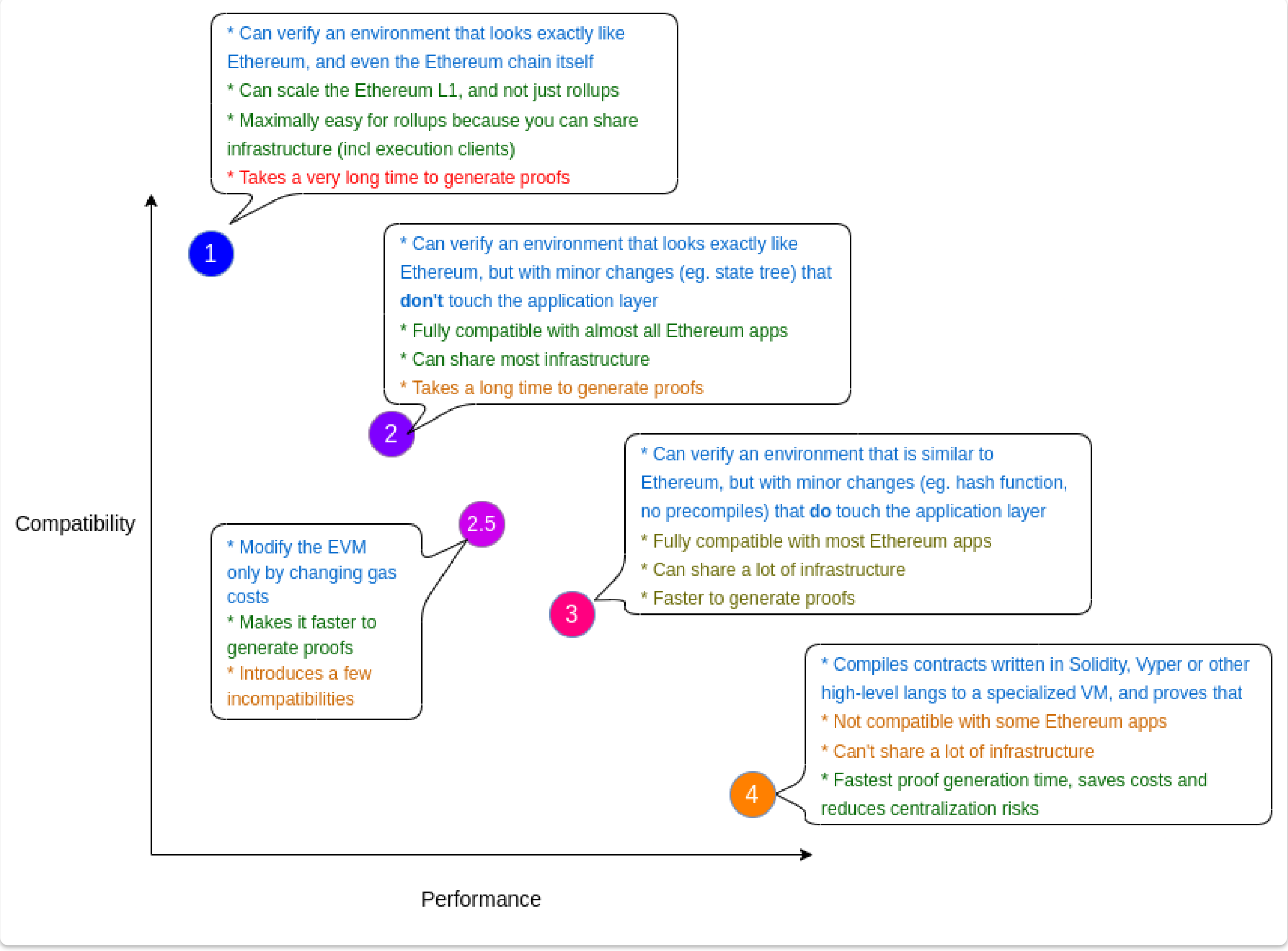
What can be done, with a signature for a particular message m and public key X is make this a valid signature for a different message m' and public key X' , but this wouldn't give us the corresponding private key.
So even though we can generate a valid signature for a particular combination of message and public key, without the corresponding private key we wouldn't be able to exploit this.

Articles about signature malleability
Derpturkey
Coder's Errand

zkEVM taxonomy



See Vitalik article



* Can verify an environment that looks exactly like Ethereum, and even the Ethereum chain itself
* Can scale the Ethereum L1, and not just rollups
* Maximally easy for rollups because you can share infrastructure (incl execution clients)
* Takes a very long time to generate proofs

* Can verify an environment that looks exactly like Ethereum, but with minor changes (eg. state tree) that **don't** touch the application layer
* Fully compatible with almost all Ethereum apps
* Can share most infrastructure
* Takes a long time to generate proofs

* Modify the EVM only by changing gas costs
* Makes it faster to generate proofs
* Introduces a few incompatibilities

* Can verify an environment that is similar to Ethereum, but with minor changes (eg. hash function, no precompiles) that **do** touch the application layer
* Fully compatible with most Ethereum apps
* Can share a lot of infrastructure
* Faster to generate proofs

* Compiles contracts written in Solidity, Vyper or other high-level langs to a specialized VM, and proves that
* Not compatible with some Ethereum apps
* Can't share a lot of infrastructure
* Fastest proof generation time, saves costs and reduces centralization risks

Type 1 (fully Ethereum-equivalent)

See zkEVM research team

Type 2 (fully EVM-equivalent)

(not quite Ethereum-equivalent)
Scroll
Hermez

Type 2.5 (EVM-equivalent, except for gas costs)

Type 3 (almost EVM-equivalent)

Scroll and Hermez in their current form

Type 4 (high-level-language equivalent)

ZKSync
Starknet + Warp