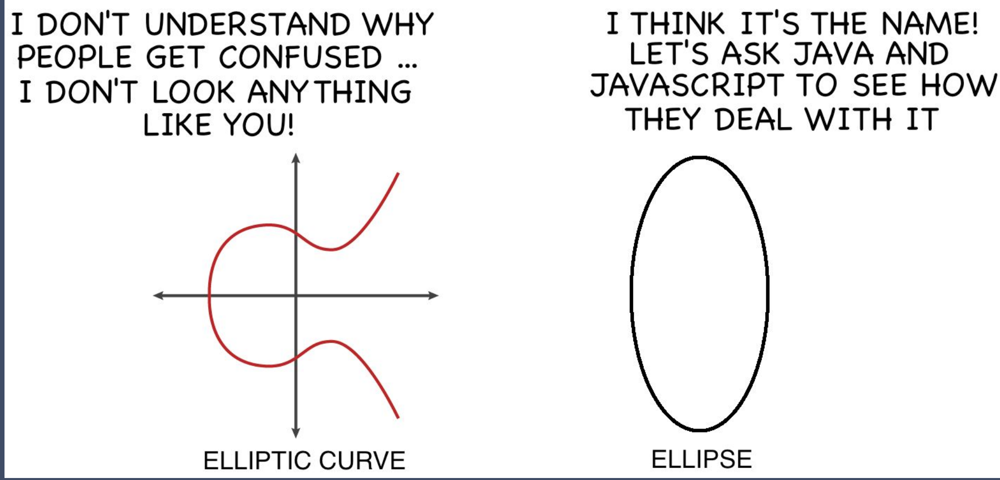
One Key To Rule Them All

Yolan Romailler
@AnomalRoil

May 2019 — North Sec

KUDELSKI SECURITY

Refresher: Elliptic curves



Seen last year at NSec !

In the talk “Getting ahead of the elliptic curve” by Martijn Grooten

An elliptic curve will simply be the set of coordinates **(x,y)** :

$$\{(x,y) \in \mathbb{F}_p^2 \quad | \quad y^2 = x^3 + ax + b, \quad 4a^3 + 27b^2 \neq 0\} \cup \{0\}$$

on which we can define an addition law “+” that is well-behaved.

Crypto refresher: ECC

ECC aka "Elliptic Curve Cryptography"

- Working with the subgroup of prime order **m** of an **elliptic curve C** generated by a given **generator G**
- Security based on the hardness of the EC discrete logarithm problem (DLP):

> If the point **Q = nP** is obtained by scalar multiplication of an integer **n** with the point **P**, being given only **Q** and **P**, it is hard to retrieve **n**.

Refresher: EC

“Scalar multiplication”?

Comes naturally from the additive operation “**+**” under which Elliptic Curves over a finite field $\mathbb{F}_p$ form an abelian group, as you’ve seen last year ;)

$$nP = \underbrace{P + \cdots + P}_{n \text{ times}}$$

By definition, scalar multiplication is naturally distributive over EC addition:

$$aP + bP = (a + b)P$$

Crypto refresher: ECC

ECC aka “Elliptic Curve Cryptography”

- Working with the subgroup of prime order **n** of an **elliptic curve C** generated by a given **generator G**.
- Security based on the hardness of the EC discrete logarithm problem.

In common EC schemes (ECDSA, ECDH, ECIES), we usually have that:

- A **private key** is simply an integer **k < p** (it is an element of $\mathbb{F}_p$)
- A **public key** is a point P = (x, y) on the curve generated by **scalar multiplication** of the private key with the base point: **P = kG**
  - where (x, y) are the coordinates of the point **on** the curve C

Using ideas coming from the Bitcoin world

Have you ever heard of the cool idea behind the BIP-32 proposal also known as "**Hierarchical Deterministic Wallets**" by Pieter Wuille?

It introduced the idea of having so-called *child keys* derived from *parent keys* when working with a EC signature algorithm such as ECDSA or Schnorr signatures.

This is made possible using the simple math on EC we just saw!

The principle behind BIP-32

In Bitcoin, addresses are **the hash of a ECDSA public key**, which is a point on an Elliptic Curve. It uses the same kind of key pair that we discussed earlier.

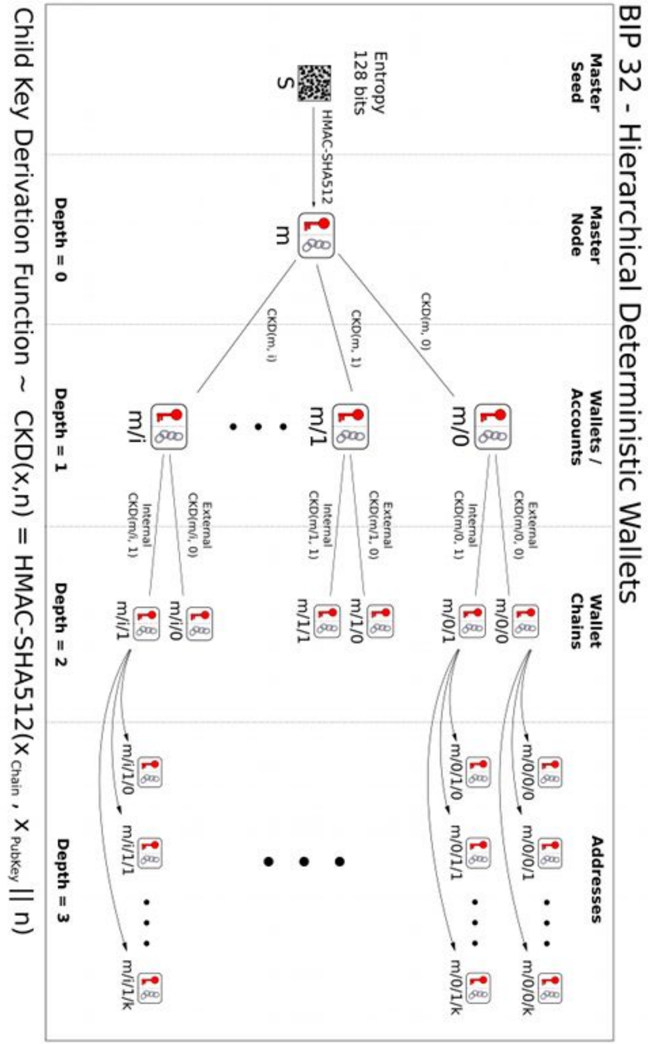
The idea is that if you have a private key **k** and a public key **P = kG**, you can pick a deterministically generated value **r** and if you take the point **Q = P + rG**, you actually just computed the public key for the private key **k + r**, since

P + rG = kG + rG = (k + r)G by distributivity !

BIP-32, the stuff I skipped

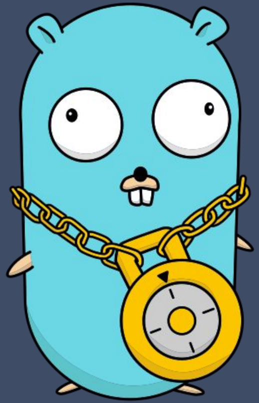
- It uses “extended” keys that contain both k and a so-called “chain code”
- It uses HMAC to derive the value **r** using the chain code as a HMAC key
- It uses indices **i** to index its child keys
- It has a notion of hardened keys that do not allow for public key derivation
- Its keys are nodes of a key tree because it enables fun bitcoin stuff

Nonetheless, the core idea is that **P + rG = (k + r)G** and that’s reusable!



Public keys… for what?

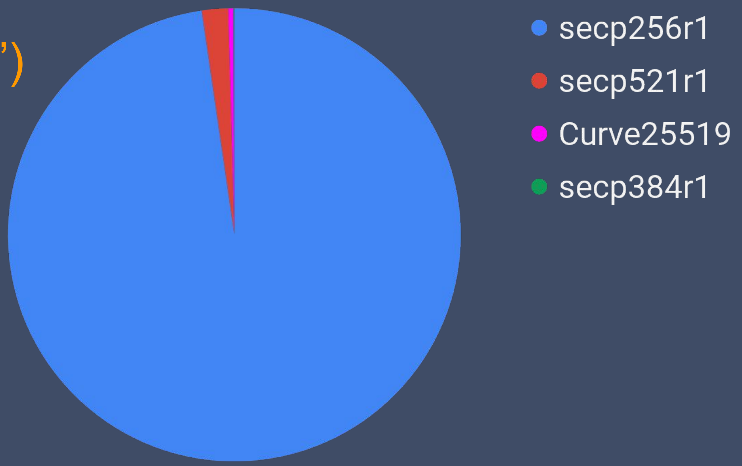
- **Connecting to remote machines without relying on ~~weak~~ mere passwords!**
- Sending PGP emails?
- Connecting to ~~cool~~ modern VPNs such as **Wireguard**.
- Sending / receiving cryptocurrencies.
- Encrypting passwords with tools such as **gopass**.
- ...



ECC SSH keys out there

Most common elliptic curves for SSH according to a world-wide key collection:

- secp256r1 97,68% (aka “NIST P-256”)
- secp521r1 1,87%
- Curve25519 0,37% (Ed25519)
- secp384r1 0,07%



Ed25519 isn’t directly compatible with BIP-32 because it hashes its private key. (Can be worked around by tweaking the signing algorithm, but not a drop-in replacement.)

What for?

SSH key derivation sounds fun, but why is it interesting?

- Anyone can generate a new public key for you being given your public key, and you'll have the private key as long as they tell you the random **r** they used
- You can generate new keys as you wish, but just need to store one single key!
- It's ~~cool~~ easy to use, and doesn't weaken security
- It might allow for fun PKI systems, where you can link a public key to another one by revealing the random value **r** shall

Demo

./1key -key path/to/key derivation-code

-a allows to export your private key (and even to encrypt it)

-q adds the key to the ssh-agent on unix devices and then quit

-rm removes ALL the keys in ssh-agent. But ~/.ssh/* keys will be reloaded.

-s “user@192.168.0.42” will “exec” the ssh command with the provided args.

Disclaimer: just a POC

But you can test it using your keys, and it works!

I have open-sourced my (go) code on:

github.com/kudelskisecurity/1key

How about security?

Having a single SSH key is not bad *per se*. However it can leak data:

- https://github.com/anomalroil.keys
- https://gitlab.com/anomalroil.keys

Using child keys is not worse than using a single private key.

It can also enable some nice “recovery” possibilities, if you loose your computer.

BTW: always store your private keys in an encrypted form!

But...

From child key to master key!

A child key is simply the master key **k** plus a given value **r**.

So, if one knows the child key **(k + r)**, and knows the value r, it is easy to recover the value **k = (k + r) - r**

This can both:

- allow you to have different private keys on each of your devices and yet a single “master public key”
- be a problem if one of your private child key is compromised and the attacker knows the value **r**

(Can be done using ./1key -i -r “hex-value”)

Applying this idea to other fields?

- Wireguard key distribution? Uses Ed25519, where the private key is hashed...
  Requires some tweaking on Wireguard itself :(
- Honest ransomware aka “RIP-001” ;)
- “Disposable keys” with “opt-in” non-repudiability (you can reveal **r** or not)

Conclusion

You can try it out, if you want. Open issues on Github, or submit PR!

Mind your keys, keep them encrypted!

Thank you!

Follow me on Twitter:
@AnomalRoil

Links

- Just a reminder to check your RSA public keys:

  https://keylookup.kudelskisecurity.com/
- Find my open source code on Github:

  https://github.com/kudelskisecurity/1key
- Find more results and analysis on our blog:

  https://research.kudelskisecurity.com
- Download these slides on my homepage (in a couple of days):

  https://romailler.ch/page/talks/