Reaping and breaking keys at scale: when crypto meets big data

Nils Amiet
Yolan Romailler

August 2018 — DEF CON 26

Warning: These slides are not up to date

Get the latest slides from:
https://research.kudelskisecurity.com/

Warning: if you want to test your keys live

You can go to our website:

keylookup.kudelskisecurity.com

and submit your key to test it against our dataset!

If the key is already known, results are immediate, otherwise you'll have to check back later.

We'll come back to this during our demo.

The problem

- Asymmetric cryptography relies on two type of keys:
  - the public key, that anybody can use to encrypt/verify data.
  - the private key, that should remain secret and allows for decryption/signing of the said data.
- Tons of public keys are out there:
  - TLS/SSL
  - SSH
  - PGP, …
- Can public keys leak data about the private keys?

Crypto recap

- RSA (Rivest–Shamir–Adleman)
  - Choose two **large prime numbers** p and q
  - Public key **(n, e)**
    - with $n = p * q$
    - and some e such that e and $\lambda(n)$ are coprime
  - Private key **(n, d)** where $d \equiv e^{-1} \pmod{\lambda(n)}$
  - Message encryption
    - $c \equiv m^e \pmod{n}$
  - Ciphertext decryption
    - $m \equiv c^d \pmod{n}$
  - RSA security relies on the hardness of the **integer factorization problem**

Crypto recap (cont'd)

- ECC ("Elliptic Curve Cryptography")
  - Security based on the hardness of the EC **discrete logarithm problem**
    - It is hard to retrieve the integer d, knowing the points G and Q=dG
  - Working with an elliptic curve C
  - Private key is an integer d
  - Public key is a point Q = (x, y) = dG
    - where (x, y) are the coordinates of the point **on** a given known curve

Passive attacks on public keys

- The Return of Coppersmith’s Attack (ROCA)
- **RSA modulus factorization** (Batch GCD)
- Invalid parameters
  - DSA generator
  - Key sizes
  - Invalid curve attacks

★ Batch GCD already used in 2010, 2012, 2016 to break weak keys
  - on datasets <100M keys

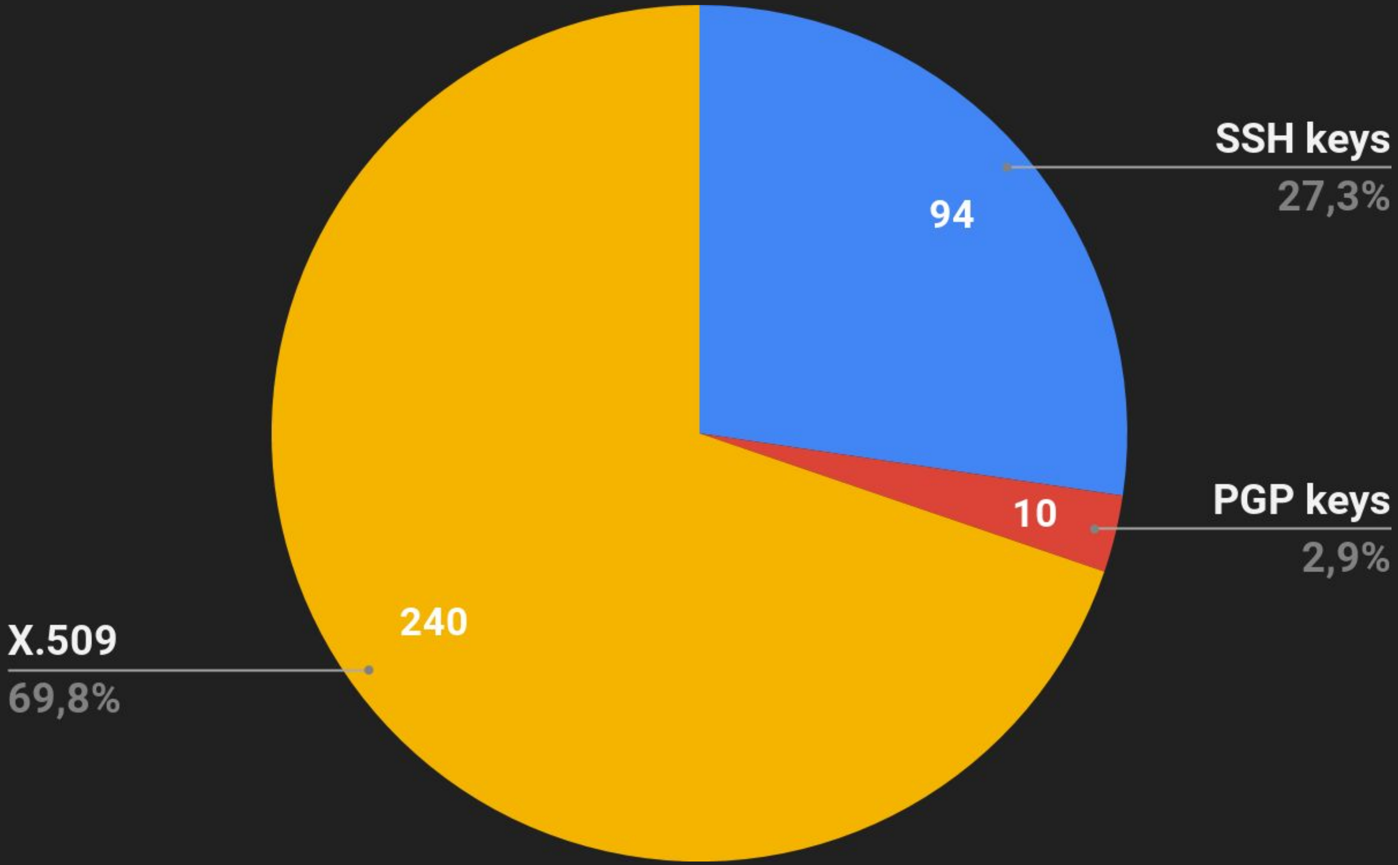
★ These are all **known attacks!**

★ And they are completely passive, the target is left unaware

Collecting public keys

- X.509 certificates
  - HTTPS, on IPv4 range and on CT domain names (>260 million)
  - IMAP(S), POP3(S), SMTP(S)
- SSH keys
  - Github.com, Gitlab.com
  - SSH host key scans (port 22)
- PGP keys
  - Github.com, Keybase.io
  - Key dumps from SKS pool of PGP key servers

Keys (millions) per key container type



Keys collected per data source

- PGP keys
  - 9.5M on SKS key servers
  - 220k on Keybase.io
  - 6k on Github.com
- SSH keys
  - 71M from CRoCS dataset
  - 17M from SSH scans
  - 4.5M on Github.com
  - 1.2M on Gitlab.com
- X.509 certificates
  - > 200M from HTTPS scans
  - 1-2M each from SMTP(S), POP3(S) and IMAP(S) scans

Our public keys stash: Big Brother style

- Attacks like RSA Batch GCD work best with larger datasets
  - More keys = more chances of finding common factors
- We collected as many public keys as we could
  - > 343,492,000 unique keys and growing
  - collection made over 1 year

Key types

● RSA	324M	324476553
● ECC	13M	13975895
● DSA	2.5M	2568969
● ElGamal	2.4M	2468739
● GOST R 34.10-2001	1k	1759
● Other	<1k	217

What to do with the (big) data

1. Collect raw data
2. Parse raw data
3. Run additional tests on parsed data
4. Ingest parsed data into database
5. Run queries on distributed database
6. ???
7. Profit

Toolbox

To collect data:

- Fingerprinting with cert/key grabbing: **Scannerl** with custom modules
- To parse it: Python code
- To ingest it: NiFi and HDFS
- To study it: Presto

To break keys:

- Batch GCD on RSA keys, using a custom distributed version
- ROCA attack on vulnerable RSA keys
- Sanity checks on EC keys

Demo

[Live demo, using https://keylookup.kudelskisecurity.com/]

[Factoring a key with the GCD, rebuilding the private key out of the factors]

Behind the scenes

- Distributed fingerprinting using 50 Scannerl slaves
- Batch-GCD:
  - 280 vCPUs cluster across 7 nodes
  - 1.4 TB storage for intermediate results
- Data Lake with 10+ data nodes
- Storage: over 15TB including all raw data

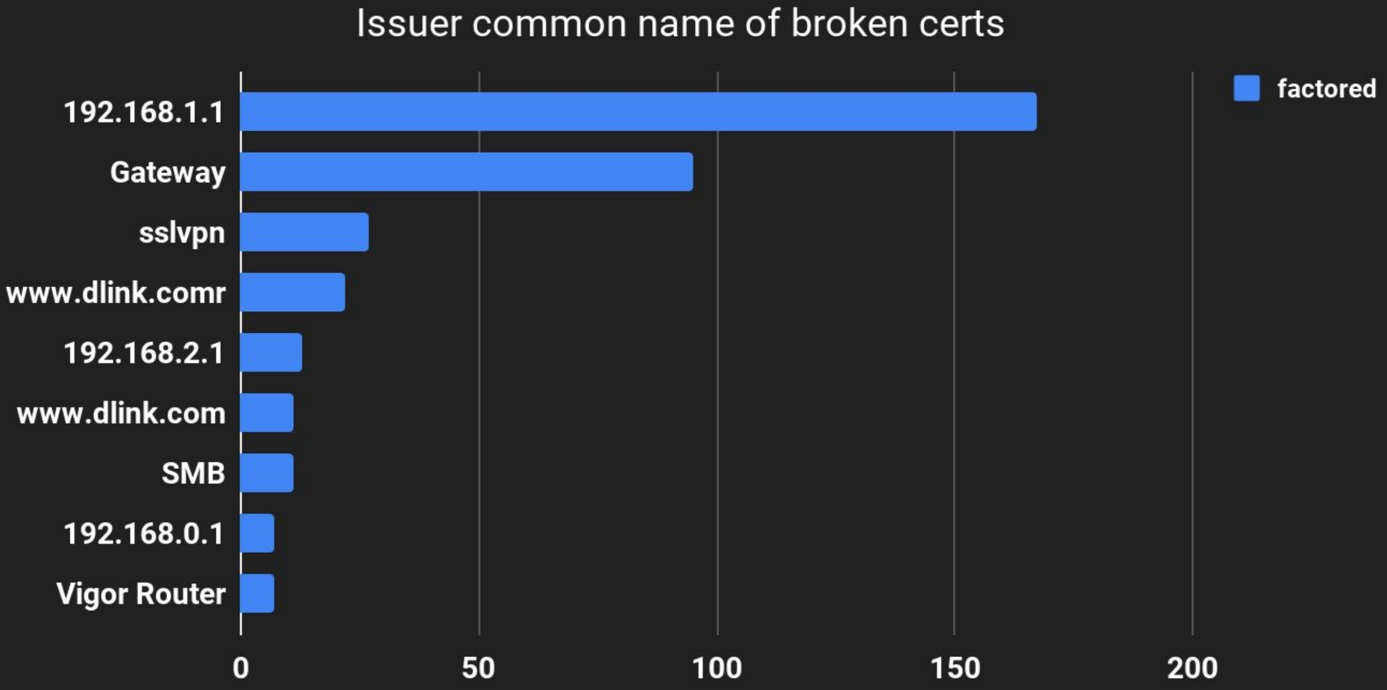
Results: RSA keys

Broken keys:

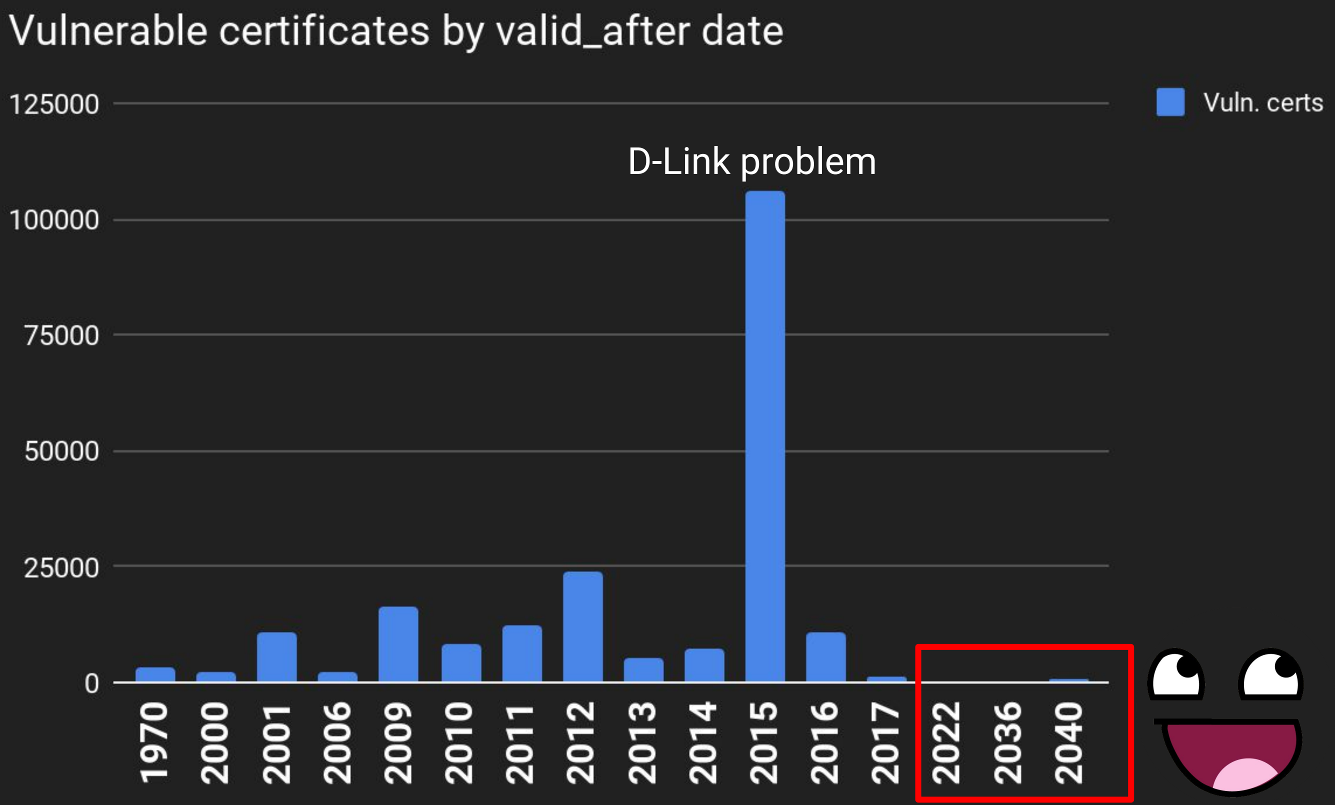
- >4k RSA keys vulnerable to ROCA
  - 33% of size 2048 (weak), 64% of size 4096 (should be fine)
  - Mostly PGP keys (97%)
  - Found vulnerable keys on Keybase.io, Github.com and Gitlab.com! Check your keys!
- >200k RSA keys factored through batch GCD
  - Real-world breakable/broken keys!
  - 207k X.509 certificates, allowing for MitM attacks
    - at least 261 certs currently in use, 1493 certs used over last year
  - >1200 SSH keys, allowing for MitM attacks
  - 6 PGP keys, allowing for decryption, impersonation and other evil

Results: RSA keys

Unsurprisingly, many routers are concerned:



Results: RSA keys

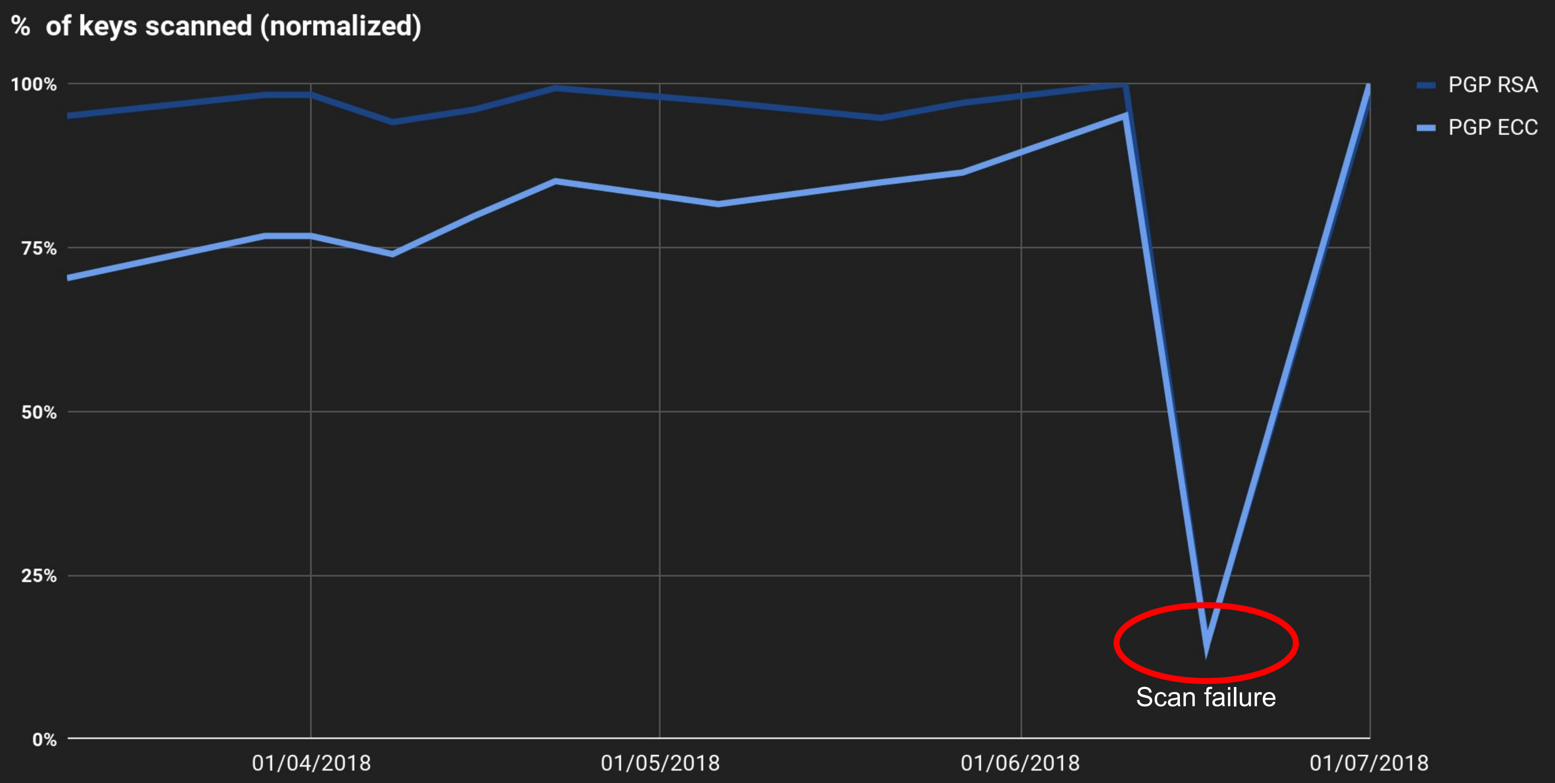


Results: ECC keys

- The adoption rate of ECC differs greatly depending on the source:
  - X509 and PGP are steadily adopting ECC
- Most common curves for SSH:

secp256r1	97,68%
secp521r1	1,87%
Curve25519	0,37%
secp384r1	0,07%

Growth of ECC keys



Fun facts

- Some keys are used as both PGP keys, SSH keys and/or X509 certs!
- PGP subkey/master key ratio
    - 50.5% master keys
    - 49.5% subkeys
    - Most people have only one subkey?!
- At least 361 keys we could factor had more than 2 factors!
- DSA is dead (OpenSSL deprecated it in 2015):
    - only 3106 X.509 certs seen over last year
    - < than 0.55% of SSH keys are DSA based

Conclusion

- Mind your keys!
- Anybody can do the same kind of silent attack!

  And maybe they already do...
- Find our open source code on Github
  - https://github.com/kudelskisecurity
  - https://github.com/kudelskisecurity/scannerl
- Find more results and analysis on our blog
  - https://research.kudelskisecurity.com