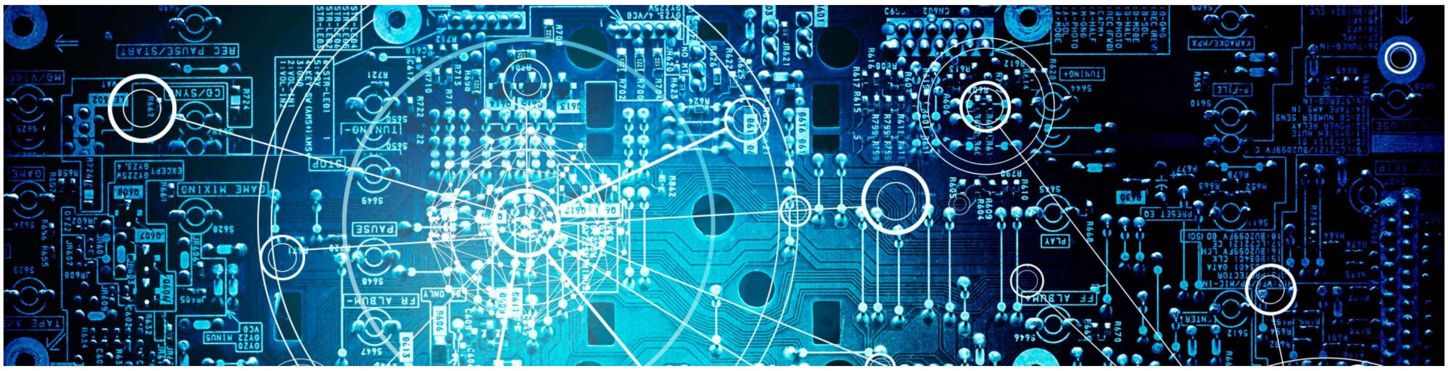


REAPING AND BREAKING KEYS AT SCALE: WHEN CRYPTO MEETS BIG DATA

October 16, 2018 Yolan Romailler Conferences and events, Crypto, Research Leave a comment

Co-authored by Nils Amiet and Yolan Romailler

SEARCH

Se

CATEGORIES

Select

ARCHIVES

Select

This is a summary of the talk we gave on the 11th of August at DEF CON 26 in Las Vegas.

Our slides are available here.

This research was about public keys found on the internet. As you know, public keys are meant to be publicly available in order to typically be able to verify signatures produced by the private key holder or to encrypt content that only the private key holder is able to decrypt.

One question we may ask ourselves is: to which extent can we deduce the private keys out of their respective public keys?

Unsurprisingly, the theoretical answer is none! It is not possible at all to recover the private key out of a public key without solving problems that are considered intractable with technology at our disposal today.

Whew! We are safe, aren't we?

Well… The thing is: there is the theoretical answer, which is even sometimes mathematically proven to be absolutely true, but there is also the "practical" answer when we take into account that the theory might not have been translated perfectly into code and that implementations may have some flaws.

So we decided to gather as many keys as we could and see how is the practical state of public key cryptography on the internet.

Crypto recap

When we are talking about public key cryptography used in practice, we are generally thinking about the following cryptographic schemes:
– the RSA schemes (named after its creators, Rivest, Shamir and Adleman), which are based on the hardness of the integer factorization problem and allow to do both public key encryption and digital signature.
– the DSA scheme (aka Digital Signature Algorithm), based on the hardness of solving the discrete logarithm problem in specific groups. Note that DSA has been deprecated a few years ago by OpenSSH and allows to do only, as its name implies, digital signatures.
– some ECC schemes, (aka Elliptic Curve Cryptography), mostly ECDSA for digital signature, and ECDH for key exchange.

RSA reminder

Generating an RSA keypair involves choosing two large prime numbers p and q and computing the public modulus n which is the product of p and q. In other words $n = p \cdot q$.

RSA's security relies on the fact that it is hard to retrieve the prime factors p and q given only n. This is known as the integer factorization problem.

In order to retrieve p and q from n, an attacker could try and divide n by all prime numbers until they find one for which the result of the division is an integer. In practice this is not feasible due to the time it would take to try all possibilities, because the primes are typically integers of 2048 bits, which are way too big for a brute-force approach.

Therefore the public key can safely be shared with everyone without compromising the security of the cryptosystem.

ECC reminder

Elliptic curve cryptography is generally relying on the hardness of the discrete logarithm problem.
It implies specifying an elliptic curve C, with a generator G, which are both public.
Then a key pair can be generated by taking a large integer d as the secret key and generating the public key point $Q = (x, y) = d \cdot G$, where (x, y) are the coordinates of the point corresponding to the public key on the curve C.
It is crucial that the public key point Q actually lies on the curve C.

Passive attacks on public keys

Some well-known public key attacks include The Return of Coppersmith's Attack (ROCA), which exploits a flaw in Infineon's RSALib, a library that generated RSA keys and which was incorporated in many smartcards. Keys of size 2048 bits or less that were generated with this library are considered vulnerable to ROCA.

In addition to ROCA, public keys can be generated with invalid parameters or even malicious parameters… For instance, DSA and ECDSA are quite sensible to invalid parameters, but having a key length considered too small by current security standards can also be a problem.

The Batch-GCD attack, which was already used in 2012 (Lenstra et al. and Heninger et al.) and 2016 (Hastings et al.), can in practice help retrieve private keys from public keys by computing the greatest common divisor between all public key modulus pairs in a public key dataset. Whenever the GCD between two moduli is larger than one, we have found a factor common to both keys in the considered pair, which we can leverage to retrieve both private keys, as long as these are common RSA keys. Indeed, this attack doesn't work fully for multi-prime RSA since you would still need to factor the quotient of the division of the modulus by the GCD in order to retrieve the full private key. However, it still significantly reduces the security margin of a multi-prime RSA key whenever a common factor is found.

Collecting public keys

We targeted 3 key container types in our project:

- X.509 Certificates
- SSH keys
- PGP keys

We were able to collect over 240M unique keys from certificates, 94M SSH keys, including both host keys and also user keys. We also collected 10M PGP keys mainly from keyserver dumps. More details below.

Keys collected per data source

- X.509 certificates
  - Over 200M from HTTPS scans
  - 1-2M each from SMTP(S), POP3(S) and IMAP(S) scans
- SSH keys
  - 71M from CRoCS dataset
  - 17M host keys from SSH scans
  - 4.7M on Github.com
  - 1.2M on Gitlab.com
- PGP keys
  - 9.5M on SKS key servers
  - 220k on Keybase.io
  - 8k on Github.com

Key types

RSA remains the most used public key algorithm with over 90% of all keys. Elliptic curve keys are the follow-up and finally, there are some more exotic key algorithms, such as DSA, ElGamal and GOST R, to name a few. More details in the table below.

Public key algorithm	**Number of keys collected**	**Percentage**
RSA	327M	94.48%
ECC	14M	4.05%
DSA	2.6M	0.75%
ElGamal	2.5M	0.72%
GOST R 34.10-2001	1k	~0%

Public key algorithm	Number of keys collected	Percentage
Other	Less than 1k	~0%

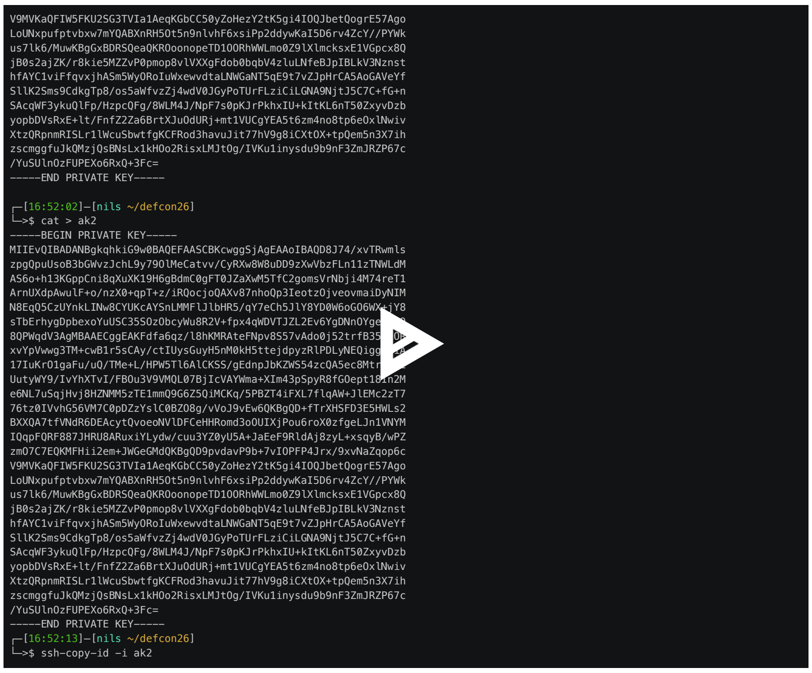
Tools

We have been collecting X.509 certificates and SSH host keys with Scannerl. Keys from other sources, such as Gitlab.com, Github.com or Keybase.io are collected using mostly Python scripts, known as k-reaper, which are now open source. We use Apache NiFi to ingest parsed data into a database stored on HDFS. We use partitioned Presto tables to quickly explore our dataset.

We break RSA keys using a custom distributed implementation of Batch-GCD written in Chapel. We also wrote Python and Go scripts for verifying vulnerability to ROCA and ECC sanity checks.

GCD demo with 2 keys

You can watch the following screencast for an example of how easily common factors are retrieved and lead to private key recovery:



```
V9MVKaQFIW5FKU2SG3TVIa1AeqKGbCC50yZoHezY2tK5gi4IOQJbetQogrE57Ago
LoUNxpufptvbxw7mYQABXnRH5Ot5n9nlvhF6xsiPp2ddywKaI5D6rv4ZcY//PYWk
us7lk6/MuwKBgGxBDRSQeaQKROoonopeTD1OORhWWLmo0Z9lXlmcksxE1VGpcx8Q
jB0s2ajZK/r8kie5MZZvP0pmop8vlVXXgFdob0bqbV4zluLNfeBJpIBLkV3Nznst
hfAYC1viFfqvxjhASm5WyORoIuWxewvdtaLNWGaNT5qE9t7vZJpHrCA5AoGAVeYf
SllK2Sms9CdkgTp8/os5aWfvzZj4wdV0JGyPoTUrFLziCiLGNA9NjtJ5C7C+fG+n
SAcqWF3ykuQlFp/HzpcQFg/8WLM4J/NpF7s0pKJrPkhxIU+kItKL6nT50ZxyvDzb
yopbDVsRxE+lt/FnfZ2Za6BrtXJuOdURj+mt1VUCgYEA5t6zm4no8tp6eOxlNwiv
XtzQRpnmRISLr1lWcuSbwtfgKCFRod3havuJit77hV9g8iCXtOX+tpQem5n3X7ih
zscmggfuJkQMzjQsBNsLx1kHOo2RisxLMJtOg/IVKu1inysdu9b9nF3ZmJRZP67c
/YuSUlnOzFUPEXo6RxQ+3Fc=
-----END PRIVATE KEY-----

┌─[16:52:02]─[nils ~/defcon26]
└─>$ cat > ak2
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQD8J74/xvTRwmls
zpgQpuUsoB3bGWvzJchL9y79OlMeCatvv/CyRXw8W8uDD9zXwVbzFLn11zTNWLdM
AS6o+h13KGppCni8qXuXK19H6gBdmC0gFT0JZaXwM5TfC2gomsVrNbji4M74reT1
ArnUXdpAwulF+o/nzX0+qpT+z/iRQocjoQAXv87nhoQp3IeotzOjveovmaiDyNIM
N8EqQ5CzUYnkLINw8CYUKcAYSnLMMFlJlbHR5/qY7eCh5JlY8YD0W6oGO6WX+jY8
sTbErhygDpbexoYuUSC35SOzObcyWu8R2V+fpx4qWDVTJZL2Ev6YgDNnOYge[illegible]
8QPWqdV3AgMBAAECggEAKFdfa6qz/l8hKMRAteFNpv8S57vAdo0j52trfB35[illegible]
xvYpVwwg3TM+cwB1r5sCAy/ctIUysGuyH5nM0kH5ttejdpyzRlPDLyNEQigg[illegible]
17IuKrO1gaFu/uQ/TMe+L/HPW5Tl6AlCKSS/gEdnpJbKZWS54zcQA5ec8Mtr[illegible]
UutyWY9/IvYhXTvI/FBOu3V9VMQL07BjIcVAYWma+XIm43pSpyR8fGOept18[illegible]n2M
e6NL7uSqjHvj8HZNMM5zTE1mmQ9G6Z5QiMCKq/5PBZT4iFXL7flqAW+JlEMc2zT7
76tz0IVvhG56VM7C0pDZzYslC0BZO8g/vVoJ9vEw6QKBgQD+fTrXHSFD3E5HWLs2
BXXQA7tfVNdR6DEAcytQvoeoNVlDFCeHHRomd3oOUIXjPou6roX0zfgeLJn1VNYM
IQqpFQRF887JHRU8ARuxiYLydw/cuu3YZ0yU5A+JaEeF9RldAj8zyL+xsqyB/wPZ
zmO7C7EQKMFHii2em+JWGeGMdQKBgQD9pvdavP9b+7vIOPFP4Jrx/9xvNaZqop6c
V9MVKaQFIW5FKU2SG3TVIa1AeqKGbCC50yZoHezY2tK5gi4IOQJbetQogrE57Ago
LoUNxpufptvbxw7mYQABXnRH5Ot5n9nlvhF6xsiPp2ddywKaI5D6rv4ZcY//PYWk
us7lk6/MuwKBgGxBDRSQeaQKROoonopeTD1OORhWWLmo0Z9lXlmcksxE1VGpcx8Q
jB0s2ajZK/r8kie5MZZvP0pmop8vlVXXgFdob0bqbV4zluLNfeBJpIBLkV3Nznst
hfAYC1viFfqvxjhASm5WyORoIuWxewvdtaLNWGaNT5qE9t7vZJpHrCA5AoGAVeYf
SllK2Sms9CdkgTp8/os5aWfvzZj4wdV0JGyPoTUrFLziCiLGNA9NjtJ5C7C+fG+n
SAcqWF3ykuQlFp/HzpcQFg/8WLM4J/NpF7s0pKJrPkhxIU+kItKL6nT50ZxyvDzb
yopbDVsRxE+lt/FnfZ2Za6BrtXJuOdURj+mt1VUCgYEA5t6zm4no8tp6eOxlNwiv
XtzQRpnmRISLr1lWcuSbwtfgKCFRod3havuJit77hV9g8iCXtOX+tpQem5n3X7ih
zscmggfuJkQMzjQsBNsLx1kHOo2RisxLMJtOg/IVKu1inysdu9b9nF3ZmJRZP67c
/YuSUlnOzFUPEXo6RxQ+3Fc=
-----END PRIVATE KEY-----
┌─[16:52:13]─[nils ~/defcon26]
└─>$ ssh-copy-id -i ak2
```

And if you want to know whether your key is vulnerable to batch-gcd, you can go to our keylookup.kudelskisecurity.com website and check it today!

Notice that these are public keys, and if your public key has been generated correctly, by a non-faulty implementation, you are not putting yourself at risk by submitting your public key to our service. Normally, a well-generated key has such a negligible chance of having a common factor with any other well-generated key, that the probability that it occurs is far smaller than that of winning the lottery, or even that of getting hit by a meteorite!

Infrastructure

In order to perform the key collection, we were able to leverage our custom, in-house, open-source fingerprinting engine “Scannerl”! This was done by developing Scannerl modules, which are now open-sourced, in order to collect certificates whenever we fingerprint a compatible server in front of us. The sources currently supported are HTTPS certificates, IMAP(S), POP3(S), SMTP(S) and SSH host keys.

We used 50 Scannerl slaves for every scans, in order to cover the whole IPv4 range, as well as a significant number of domain names (over 270M) we had collected from Certificate Transparency log servers to perform our scans. Notice that now, thanks to this project, Scannerl supports SNI.

In order to run our distributed, incremental version of the batch-gcd algorithm, we leveraged a 280 vCPUs cluster and we needed over 2 TB of storage just to store the product trees that are being built when running batch-gcd!
The big novelty of our algorithm is that we can process keys incrementally, that is, we do not need to re-run the batch-gcd algorithm over the whole dataset, but we can instead simply process the new keys using the previously stored results to compute the GCD of all newly-added keys relatively to all already-known keys!

The results were then stored in our HDFS cluster with over 10 data nodes, and we could achieve quick database lookups on the whole dataset thanks to so-called partitioned tables, enabling us to give an answer to your queries on our keylookup.kudelskisecurity.com website within seconds!

Results

So, how many keys are trivially vulnerable to a batch factorization?
We've run our custom distributed batch-gcd algorithm on our dataset made of over 340 million keys and have found that on average, 1 key out of 1600 is vulnerable to batch-gcd in our dataset. We broke over 210k RSA keys in total. Out of these vulnerable keys, 207k are X.509 certificates, most of which are not in use anymore, however over 260 certificates are still in use and over 1400 certificates are no older than 2017.

We broke Over 3100 SSH keys. We found 295 batch-gcd vulnerable PGP keys, out of which 287 have a composite factor. We found that batch-gcd vulnerable keys with composite factors were mostly PGP keys. There are more of those from PGP key sources than from SSH and X.509 key sources combined. In total, we found 486 vulnerable keys with a composite factor.

Over 4000 RSA keys are vulnerable to ROCA in our dataset. Some of those come from Keybase.io, Github.com, and Gitlab.com. 33% of those vulnerable keys have a size of 2048 bits and are therefore considered weak. 64% of those ROCA-vulnerable keys are of size 4096 bits and are therefore much harder to break and considered safe, even though the ROCA test is positive.

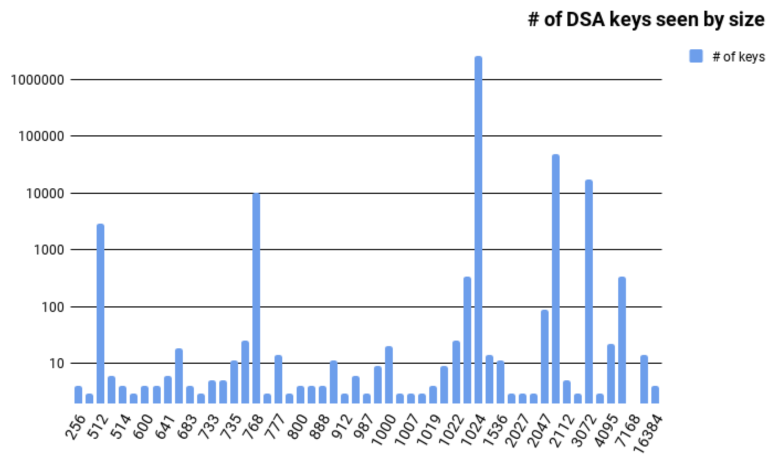
Many batch-gcd vulnerable X.509 certificates are used on routers or home gateways. We considered vulnerable certificates that were issued in 2017 or later (only the recent ones) and were able to see the most frequent Issuer Common Names for those (see slide 24 in our presentation). We also found batch-gcd vulnerable certificates that have never been valid yet. In other words, they will start being valid in the future. This is quite concerning and we wonder why anyone would do such a thing, knowing that in the future an RSA key generated with parameters considered safe today may just become easy to break so why generate the certificate today if it’s not to be used immediately while you could just wait until you need to use it and decide on parameters at that time.

Another area we have investigated is the usage of ECC keys. We found that ECC X.509 certificates and PGP keys are becoming more and more frequent. We found that the most used curve for ECC SSH keys is secp256r1 (97%), followed by secp521r1 (1.87%). Finally, Curve25519 (0.37%) and secp384r1 (0.07%) are currently the least used curves for SSH keys in the wild. We expect Curve25519 to become more popular in the future and may catch up to 2nd place soon.

We found keys that are re-used in multiple key containers. In our dataset, at least 3442 keys are re-used as either PGP keys, SSH keys or X.509 certificates. That means some people have generated, for example, an X.509 certificate, have then turned their certificate into a PGP key and have made both publicly available.
We do not recommend re-using key material more than strictly necessary.

Speaking of PGP keys, we found that most people have surprisingly only one subkey in their PGP key.

Finally, DSA can officially be declared dead. It was already deprecated in OpenSSL in 2015 and we have seen only 3106 X.509 certificates with DSA keys over the last year. Additionally, less than 0.55% of all SSH keys we collected are DSA based. We also found lots of DSA keys with key sizes that are not specified in FIPS 186-3:



Recommendations

In the end, considering the results of our research and the current state of the art, we recommend using Elliptic-curve based cryptography whenever you have the choice over RSA, since it is easier to implement, more resilient to failures, and has better performances than RSA. The most advanced and secure tools that are being built nowadays are generally using such cryptographic primitives, as it is the case for Wireguard, which we discussed in a previous blog entry.

Furthermore, we advise you to use 4096 bit keys for RSA, whenever you are mandated to work with RSA.
We do not exactly recommend using multi-prime RSA, however it has proved to be better

performance-wise and harder to factor (as long as you don't have too many primes, see appendix in [Silverman]), so you might consider using, for instance, 3-prime RSA, depending on your use-case (notice that multi-prime RSA is supported by OpenSSL as defined in RFC 8017).

References

- https://keytester.cryptosense.com/
- https://keychest.net/roca
- https://keylookup.kudelskisecurity.com
- https://cybermashup.files.wordpress.com/2018/08/amiet-romailler-reaping-keys-final-slides.pdf
- https://eprint.iacr.org/2012/064.pdf
- https://factorable.net/weakkeys12.conference.pdf
- https://www.cis.upenn.edu/~nadiah/papers/weak-keys/weak-keys.pdf
- https://github.com/kudelskisecurity/k-reaper
- https://github.com/kudelskisecurity/scannerl

« Open-source crypto is no better than closed-source crypto

Audit report of IOHK's Icarus wallet »