Factoring RSA Keys Found in Certificate Transparency Logs

Håvard Raddum[1] and Henry Faltin Våge[2]

[1]Simula UiB, Norway
[2]Intility, Norway

Introduction TLS is the most important protocol for securing web communication today. Authenticating servers on the internet relies on PKI, and the number of public key certificates issued yearly by certificate authorities (CA) are now in the billions. Most of the public keys attested by these certificates are 2048-bit RSA keys [2]. The security of RSA is based on the hardness of factoring $N = pq$ where N is public and p and q are two secret big primes. With so many keys generated for certificates each year it is important that all CAs use cryptographically strong random number generators (RNG) when selecting their prime numbers. If the same prime is used in two different RSA keys N_1, N_2 it is easy to factor both by computing their greatest common divisor (GCD).

Experiments on computing the GCDs of all pairs from a large set of RSA keys have been done before. In 2012, two teams of researchers independently harvested approximately 6 million certificates with RSA keys each, and ran GCD on all pairs [4, 6]. One finding was that around 0.5% of the RSA keys could be factored this way, mostly due to faulty RNG's. The same experiment was repeated in 2016, this time collecting 81 million keys [3]. The result was that 0.4% of the RSA keys could be factored via GCD. Finally, in 2019 approximately 75 million certificates with RSA keys were collected from the internet and almost 0.6% of them were factored by taking GCD of all pairs [5]. According to the authors this is due to IoT devices with poor RNGs. The authors compare this to a set of 100 million RSA keys collected from the Certificate Transparency (CT) logs [2], where they could only factor 5 keys.

Our contribution: We have redone the GCD factoring experiment using 159 million certificates from the CT log Argon from 2021, and report on our findings. To our knowledge, this is the largest experiment of this type done so far.

Method used in factoring experiment The problem to be solved is to calculate the GCD of every pair of the numbers $N_1, \ldots, N_n$ where n is large. A naive approach of looping trough every pair would use $\mathcal{O}(n^2)$ runtime. A more efficient algorithm was written and published in [4] running in $\mathcal{O}(nlog(n)log(log(n)))$. The underlying idea is that if you multiply all numbers on the list to one large product you only need to do n GCD operations and n divisions. When $P = \Pi_{j=1}^{n} N_j$, then

$$GCD(N_i, \frac{P}{N_i}), \text{ where } 1 \leq i \leq n \tag{1}$$

will immediately tell you if N_i shares a prime with at least one of the other N_j's, and what that prime is. However, since these GCD and division operations would be done with a very large number P this approach also has too long runtime when n is large.

The team in [4] therefore refined the idea and implemented a new GCD algorithm using a product and a remainder tree. The algorithm indeed works by multiplying all the input numbers into a large product P, but P is only used in a few computations. Later computations are done on numbers whose bit-size is halved at every level in a remainder tree, see Figure 1 for a sketch of the idea. Now the algorithm can handle larger input sizes but still do not scale well enough when, say, $n \approx 10^8$. With that large input size, the problem of too high complexity trying to work with P still remains.

The bottleneck is still the large product, so in 2016 the authors of [3] redesigned the algorithm for use

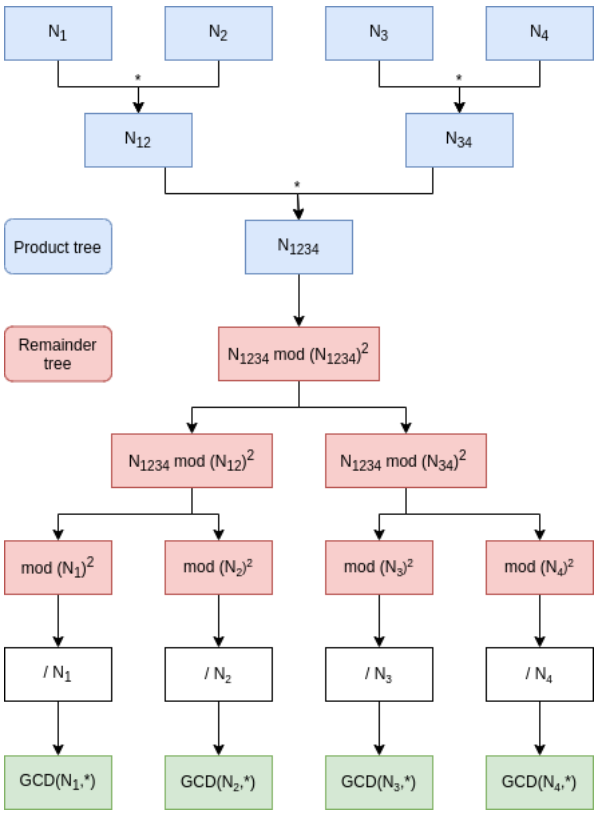

Figure 1: The original algorithm from 2012 using one product and remainder tree to compute the GCD. The idea resembles the naive one of creating a large product and then checking GCD for each number.

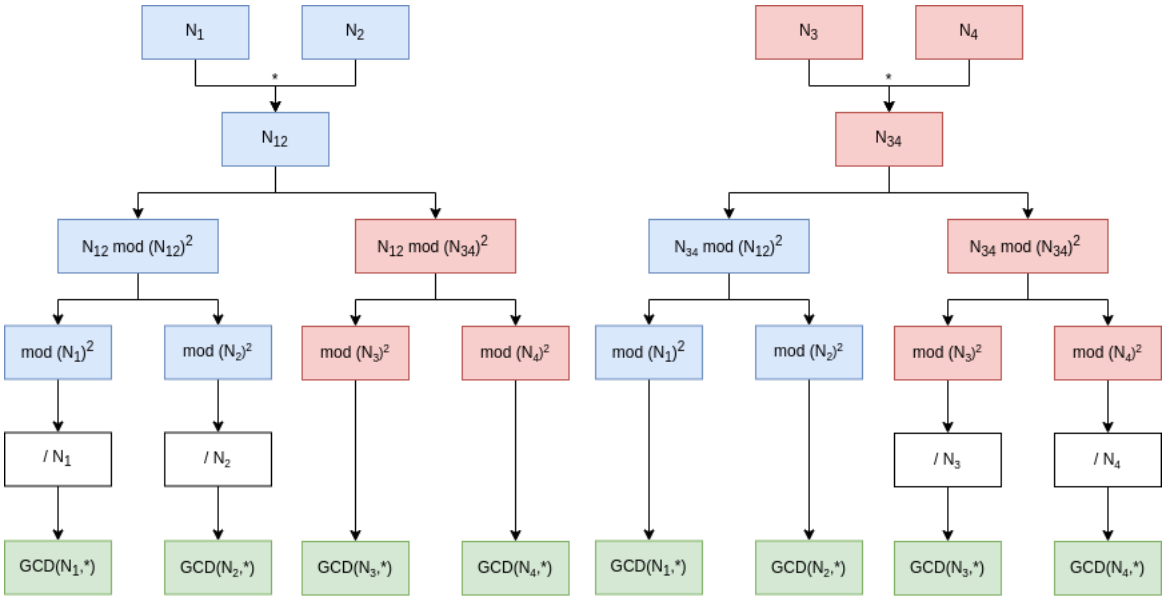

Figure 2: Product and remainder trees in Batch GCD Algorithm: With the input numbers split into k batches, the complexity of the algorithm is $O(k^2(n/k)log(n/k))$ since each batch needs to be checked against all other batches. Here we also note the difference between checking "inside" a batch and checking across batches. In our implementation, the large products N_{12} and every N_i^2 would be written to file

in distributed computing. Instead of using just one product and remainder tree, they used several, thus reducing the size of the large product into smaller chunks, or batches, as seen in Figure 2. We implemented the batch GCD algorithm from [3] to allow GCD's on smaller batches to be computed on

a single computer with multiple cores. This allowed us to use larger input data sets while still keeping the runtime practical. In addition we implemented some further improvements suggested by [5].

We collected over 159 million 2048-bit RSA keys from Google's Argon log from 2021, to our knowledge the largest set used for such an experiment. We collected these keys by running Python scripts that exclusively gathered 2048-bit RSA keys using the API defined for CT logs. To run the batch GCD algorithm some pre-processing was necessary, such as removing duplicate keys, and storing the raw moduli in files. We used a Dell server with 96 cores and 192 GB RAM to run the factoring algorithm on our data set. The algorithm finished in 88 wall clock hours running on 30 cores, peaking at 180 GB RAM and consuming over 1 TB of disk memory.

First findings Out of the 159 million unique RSA keys we were able to factor 8 different keys. This result is similar to what was found in [5], where also 1 in 20.000.000 RSA keys from the CT logs were factored. We saved the CT log index for each modulus so we could trace any factored key back to the certificate on which it appeared. When inspecting the certificates with the factored keys, we noticed they were all issued by a single CA called ZeroSSL and that they were all tied to Vietnamese domains. We then decided to do the whole experiment over again, but this time only collecting certificates issued by ZeroSSL.

Second experiment on ZeroSSL certificates We created a second script that queried the whole Argon 2021 database, which contains more than 1.35 billion certificates, searching for all certificates with 2048-bit RSA keys and having been issued by ZeroSSL. This search resulted in a set of more than 716.000 certificates. It was easy to run the GCD algorithm on the RSA moduli of this relatively small set, and it resulted in 355 unique RSA keys being factored. Going back to look at the certificates that had the broken keys, we again found the vast majority were issued to .vn domains or other top-level domains with Vietnamese-sounding names.

At this point we contacted ZeroSSL, explaining our findings. We quickly got a reply from ZeroSSL, asking for more details and informing us they would investigate internally. After a few days ZeroSSL finished their investigation and explained that the poorly generated keys were due to a single user who had "abused their system". This user had used an automated browser (like Selenium) to generate the private keys, which for some reason made the RNG weak. The answer further said that to avoid the problem a user must use a real browser and not an automated one, and ZeroSSL did not see the issue as a real vulnerability.

In [1], it is shown that different libraries generating RSA keys often leave a "signature" in the public key due to the way the random primes have been selected. The authors of [1] have also provided an online tool that identifies the library used to generate a public key, with some confidence. We submitted several of the factored keys to this tool, which told us they were certainly generated by one of the following libraries:

- Cryptix JCE 2005 03 28
- FlexiProvider
- mbedTLS version 2.8.0 or earlier
- Nettle version 2.0 or earlier
- PuTTY
- Sage
- SunRSASign

Working with CT logs Finally, we can provide some insight on how it was to work with the CT logs doing our research. First of all, the API for querying the CT logs is very limited. Every certificate has an index in a CT log, and one can only ask for certificates within a range of these indices. One will only receive a limited number of certificates if the range asked for is too large. For the Argon 2021

log the maximum number of certificates one can receive for a single request is 32. Searching through a log of 1.35 billion certificates thus takes a lot of requests and a long time.

To speed up the search for ZeroSSL certificates with RSA keys we made a script and first deployed it to 50 CPU cores, each one searching through their own 2% of the Argon 2021 database. However, this caused the server hosting the CT log to give faulty responses to the requests, making the script crash. Maybe having 50 CPU's constantly sending requests to the server felt too much like a DoS attack? Reducing the number of cores to 25 solved the problem, but it also meant it took close to 10 days traversing the whole database of certificates.

The CT logs are good resources for checking on the health of the web PKI, but it would be even better if their API could be expanded. We understand there are issues with bandwidth if one can ask for, and receive, a billion certificates in one request, but is it possible to make some kind of mechanism that would allow researchers to do SQL-like queries on a CT log? Improved functionality on the server side could make the CT logs more accessible for future research.

Conclusion In the bigger picture, our factoring experiment indicates that the way CAs select random primes for RSA keys is quite good. We only found a minor issue with one CA, which is not bad overall. Previous studies in the last decade factored many more RSA keys, indicating the CA industry has improved in this respect. Picking truly random primes is not a simple task so to avoid the single issue we found one should use well-established libraries for key generation, with sufficient randomness in all environments. One can also sidestep any potential problem with RSA keys completely by switching to algorithms based on elliptic curves that are fully supported in TLS.

One difference between our study and previous ones is that we focused solely on certificates in CT logs, while earlier experiments harvested certificates directly from the internet. There are probably many certificates that are not added to the CT logs, and a future experiment could be to run a GCD factoring experiments on a set of certificates *not* found in the CT logs to check for differences in the result.

In the future we expect that quantum-safe public key encryption algorithms will be the default choice for CAs and that RSA will be phased out. NIST has already standardized Kyber as a key encapsulation mechanism, and a few other algorithms will follow in the near future. This means that the process of making TLS quantum-safe can start, but it will probably take around a decade to finish. So in the meantime, as long as RSA remains a popular choice among CAs it still makes sense to run a GCD factoring experiment every few years.

References

[1] Petr Svenda Matus Nemec Peter Sekan Rudolf Kvasnovsky David Formanek David Komarek Vashek Matyas. The million-key question – investigating the origins of rsa public keys. In *The 25th USENIX Security Symposium (UsenixSec'2016)*, pages 893–910. USENIX, 2016.

[2] Cloudflare. Merkle Town. `https://ct.cloudflare.com/logs`.

[3] Marcella Hastings, Joshua Fried, and Nadia Heninger. Weak Keys Remain Widespread in Network Devices. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, pages 49–63, New York, NY, USA, 2016. Association for Computing Machinery.

[4] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21st USENIX Security Symposium*, August 2012.

[5] Jonathan Kilgallin and Ross Vasko. Factoring RSA Keys in the IoT Era. In *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 184–189, 2019.

[6] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Public Keys. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 626–642. Springer Berlin Heidelberg, 2012.