A look at the PGP ecosystem through the key server data

Hanno Böck

2015-03-19

Abstract

PGP-based encryption systems use a network of key servers to share public keys. These key server operate on an add only basis, thus the data gives us access to PGP public keys from over 20 years of PGP usage. Analyzing this data allows searching for cryptographic weaknesses in large scale.

I created a parser script that puts the raw cryptographic data of the PGP keys into a database. Doing this allows large scale searches for well-known vulnerabilities. DSA signatures with a duplicate k value due to bad random numbers allow the calculation of the private key. Similarly analyzing RSA keys for shared prime factors allows factoring the modulus and thus also regenerating the private key.

A small number of breakable keys due to these weaknesses were found.

1 Introduction

Internet-wide scans have revealed security issues in TLS and SSH implementations in the past. In 2010 the Electronic Frontier Foundation published the SSL Observatory [4], the first public dataset of all X.509 certificates used by HTTPS hosts on public IPv4 addresses. The SSL Observatory revealed various problems within certificates such as Extended Validation certificates which didn't comply with the rules set for such certificates. Researchers of the University of Michigan have created the tool ZMap to make such scans easier and have since then published many datasets created by Internet-wide scans [2]. These datasets have been subsequently used by researchers to reveal various vulnerabilities. These examples show that large scale analysis of cryptographic data can lead to interesting discoveries and expose real-world vulnerabilities.

PGP, introduced in 1991, is an encryption solution for E-Mails and messages. Today the free PGP alternative GnuPG is mostly used. Despite heavy criticism for its lack of usability and deprecated cryptographic technologies it remains

popular today and is likely the most widely used solution for end-to-end E-Mail encryption.

PGP key servers are public directories where users can publish their public keys. These key servers have some special properties that make them interesting for analysis. They permanently share their data, so apart from the very latest additions all active key servers have the same data. They also operate on an add only basis. That means keys can never be deleted from key servers, they can just be marked invalid (revoked). Also PGP uses a web-of-trust. That means that public keys can sign other keys to indicate that the key bellongs to the claimed owner. The signatures can also be stored on the key servers.

Due to these properties the PGP key servers are a unique source of cryptographic data. Public keys from over 20 years of PGP usage can be analyzed. Most of these keys were created either with the original PGP software or with GnuPG, but there are also some other less popular OpenPGP implementations.

2 Key Parser

Many key servers provide dumps of their data for download [1]. These data dumps are just concatenated public PGP keys. Parsing this data turned out to be challenging. GnuPG does not provide library access to low level functions that would allow parsing the data. A command line tool called pgpdump [10] allows parsing the data, but it doesn't give all the details necessary to perform the desired analysis.

Therefore I decided to write my own PGP data parser in Python. It is limited in functionality and the code quality is poor. The Python script `keyr` (abbrevation for key parser) will take PGP keys data as an input and will output MySQL statements. Using this together with the provided MySQL table preset allows building a database containing the cryptographic values of keys and signatures. Non-cryptographic data is ignored and not stored.

PGP supports different public key algorithms: RSA, DSA, El Gamal and elliptic curve based algorithms. I store DSA and El Gamal together because they are very similar.

3 DSA vulnerability on duplicate k values

DSA and ECDSA have a well-known weakness: They require the generation of a random value k during each signature generation. It is crucial that this k is always unique. If two different DSA signatures use the same k value then this results in a duplicate public r value in the signature. An attacker can detect this and use this information to calculate the private key [7]. This weakness has

been used in the past to break the protection of the PlayStation 3 [3] and to steal Bitcoins from faulty clients [9].

DSA has been very popular within the PGP ecosystem. GnuPG used DSA keys by default for a long time. Therefore searching for duplicate r values in DSA signatures seems like a worthwile idea. The key server data contains different kinds of signatures that can all be tested: Signatures to bind user ids or subkeys to a primary key, signatures to other keys, revocation signatures and more. Having all the values in a database allows us searching for duplicate r values via MySQL:

```
SELECT a.keyid, a.dsa_r, a.dsa_s, b.dsa_s, a.hash, b.hash,
c.dsa_p, c.dsa_q, c.dsa_g, c.dsa_y FROM sigs_dsa a JOIN
sigs_dsa b JOIN keys_dsa c ON a.dsa_r = b.dsa_r AND a.dsa_s
<> b.dsa_s AND a.keyid = c.keyid GROUP BY a.dsa_r INTO OUTFILE
'/tmp/dsa-duplicate-r.txt';
```

This query will give around 350 results. However when trying to calculate the private keys it turns out most of these results aren't real signatures. They are merely copies of other signatures with data errors in them. This is generally something that needs to be considered when working with the key server data: The key servers don't check the correctness of the data submitted. Therefore data transmission errors can create faulty data on the key servers.

When trying to attack the DSA signatures with duplicate r values only one key is left. This key belongs to the developer of a company called PrimeFactors (domain name primefactors.com) that is developing a commercial PGP-based encryption solution. I contacted PrimeFactors and revealed my results. This was their reply:

> "[...] our head of development, confirmed he created some test keys during early work on OpenPGP implementation, and may have left one or more on a PGP key server unintentionally. His research identified the risk you highlight, and all the key generation in our shipping product versions use the Blum-Blum-Shub generator which does not suffer from the problem you mention."

This explanation seems incoherent. The selection of the random number generator is irrelevant if it is not properly seeded. Even a very secure random number generator can result in duplicate k values. I offered PrimeFactors to test their release software, however they asked me to sign a Non Disclosure Agreement (NDA) in order to get a free test version. I declined that offer.

Given that I felt the finding of only one breakable DSA key was not that spectacular I tried further to break DSA keys by assuming a very low k value. However this didn't lead to any more breakable keys.

It could also be tried to gather signatures from public mailing list archives. This would generate a much larger corpus of signatures to test.

I had a look into the code of both GnuPG and the original PGP (older versions of it were available in source code form). Judging from the code and the comments the developers of both tools were aware of the high risk of duplicate k values and they used various methods to make sure that this can never happen.

4 RSA shared prime factors

An RSA public key consists of two values - a modulus N and an exponent e. The N value is the product of two primes called p and q. If an attacker knows p or q this can be used to calculate the private key. For the security of RSA it is therefore crucial that it is not feasible for an attacker to factorize N.

If two RSA N values share one prime factor and the other one is different then this value can be calculated using the greatest common divisor (GCD) algorithm. It is possible to use a batch GCD algorithm to attack many keys in parallel.

In 2012 two research teams independently did this. A team led by Nadia Heninger found a significant fraction of HTTPS and SSH keys to be vulnerable to this attack [5]. Most of these keys came from embedded devices and could be traced back to an early boot time entropy problem. A team led by Arjen Lenstra also performed this attack on PGP [8].

Nadia Heninger has released the source code [6] for this attack, so using our database we can easily replicate it. Most of the results are faulty keys. There are two expired keys that were already known to be broken by the Lenstra team. There is one newer key that shares a prime factor with its subkey. The mail address in the key (alice@example.com) indicates that this is not a real key. It was likely the result of some experiment, maybe someone wanted to deliberately create a breakable key.

It is unknown to me what the origin of the other two keys is and why they were broken in such a way.

5 Conclusion

Searching large scale cryptographic datasets for weaknesses can lead to interesting results. I have published the source code of my key parser tool under a CC0 license to allow others to re-use my work for further analysis.

Based on my research it seems that over a very long time the use of PGP implementations with deeply flawed random number generation functions was

very rare. This is good news.

Source code

The tools used to parse the keys and attack the DSA keys will all be published on github:

`https://github.com/hannob/pgpecosystem/`

References

[1] Keydump sources. `https://bitbucket.org/skskeyserver/sks-keyserver/wiki/KeydumpSources`.

[2] ZMap Team at the University of Michigan. Internet-wide scan data repository. `https://scans.io/`.

[3] fail0verflow. 27C3 - console hacking 2010 - PS3 epic fail, 2010. `http://events.ccc.de/congress/2010/Fahrplan/events/4087.en.html`.

[4] Electronic Frontier Foundation. SSL observatory, 2010. `https://www.eff.org/observatory`.

[5] 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. `https://factorable.net/paper.html`.

[6] Nadia Heninger and J. Alex Halderman. fastgcd 1.0 source code. `https://factorable.net/resources.html`.

[7] Nate Lawson. DSA requirements for random k value, 2010. `http://rdist.root.org/2010/11/19/dsa-requirements-for-random-k-value/`.

[8] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Ron was wrong, Whit is right. Cryptology ePrint Archive, Report 2012/064, 2012. `http://eprint.iacr.org/2012/064`.

[9] Filippo Valsorda. Exploiting ECDSA failures in the Bitcoin blockchain, 2014. `https://conference.hitb.org/hitbsecconf2014kul/materials/D1T1%20-%20Filippo%20Valsorda%20-%20Exploiting%20ECDSA%20Failures%20in%20the%20Bitcoin%20Blockchain.pdf`.

[10] Kazu Yamamoto. pgpdump. `http://www.mew.org/~kazu/proj/pgpdump/en/`.