*Remembering to Mine Your Ps and Qs: An Analysis of Weak Keys and Malfunctioning Random Number Generators*

Dustin Holloway   
*Faculty of Engineering*  
*University of Ottawa*Ottawa, Canada  
dholl086@uottawa.ca  
  
David Knox  
*Faculty of Engineering*   
*University of Ottawa*Ottawa, Canada  
dknox@uottawa.ca

*Abstract*—Analysis of public keys currently in use on the public internet shows that there is a high prevalence of repeated keys across both TLS and SSH hosts (60.6% of TLS public keys and 28.6% of SSH). In addition, private keys were obtained for 134,197 TLS hosts and 108,653 SSH hosts. Lastly, vulnerable devices were identified across 55 unique device vendors. These findings support the body of previous work showing the practical issues when implementing entropic keys and demonstrate that there are significant vulnerabilities still present on the internet, including many weak and repeated keys.

Keywords—public keys, random number generation, entropy, vulnerabilities, secure communication, RSA encryption

# Introduction

Public key encryption is fundamental for authentication and confidentiality on the internet. In 2012, Heninger *et al.* conducted a comprehensive internet scan of TLS and SSH servers and found a high prevalence of insecure random number generators in use at the time, resulting in significant RSA and DSA key vulnerabilities [1]. Since the publication of this study, the growth in the number of “internet of things” devices as well as the introduction and proliferation of virtual machines has likely increased the problems of the past (e.g. shared keys, lack of entropy and vulnerable devices) despite improvements to network and device security.

A comprehensive internet scan of transport layer security (TLS) and secure shell (SSH) servers was conducted, with the goal of identifying problems with entropy and randomness as well as vulnerable hosts and devices [1]. In total, the authors collected 5.8 million unique TLS certificates and 6.2 million unique SSH host keys. They found that “0.75% of TLS certificates shared keys due to insufficient entropy during key generation… and another 1.70% [were suspected to have] come from the same faulty implementations and may be susceptible to compromise” [1, p. 1].

In addition, they also found that 0.50% of the TLS hosts and 0.03% of SSH hosts shared common prime factors, which they attributed to problems with entropy and insufficient randomness. Lastly, they identified and examined vulnerable devices and software from 54 manufacturers, concluding that the identified problems generally stemmed from “defective implementations that generate keys without having collected sufficient entropy” [1, p. 2]. Collectively, these findings demonstrated a high prevalence of insecure random number generators in use at the time, which resulted in significant Rivest-Shamir-Adleman (RSA) and Digital Signature Algorithm (DSA) key vulnerabilities. The analysis of random number generation in [1] demonstrated that it was possible to “mine for vulnerabilities and detect problems in dozens of different devices and implementations in a single shot” [p. 20] rather than rely on reverse engineering.

As of 2019, there was an estimated 14.2 internet-connected devices across the world [2], with more than 127 new devices connecting to the internet every second [3]. Virtual machines are now more common than when this study was performed, making entropy errors easier to make [4]. Given these factors, it is possible that the problems of shared keys, lack of entropy and vulnerable devices, are likely to have increased since 2012. However, the parallel rise in network and device security might be expected to mitigate this effect (e.g., most devices now use larger keys (e.g. follow a 2048 or higher RSA bit encryption standard) [5].

The goal of this study is to analyze the public keys currently in use on the public internet. A total of 38.4 million TLS public keys and 14.1 million SSH public keys were collected and analyzed. These were then analyzed todetermine the extent to which the findings in [1] remain valid, almost a decade later.

# Background

Public key encryption is used for secure communication across the internet. There are two primary types of public key protocols: TLS and SSH. TLS is the main protocol for secured encryption on the internet, and is used in HTTPS communication, while SSH is used largely to access files and facilitate other application layer protocols.

RSA is the primary method for securing TLS and SSH public keys. An RSA [6] public key is comprised of a modulus *N* and a public exponent *e*, in which the modulus *N* is the product of two random prime numbers *p* and *q.* The private key is the decryption exponent *d*, which can be calculated [1] by:

*d* = *e*-1 mod (*p* – 1)(*q* – 1)

As seen above, central to public key encryption is the generation of random numbers. When the modulus *N*, or one or both prime numbers (*q* or *p*) are known, it is possible to compute the private key for any public key. To prevent against such factorization, the U.S. Department of Commerce’s National Institute of Standards and Technology (NIST) recommends in [7] that public keys be: unique; long (minimum of 2,048 bit length); and random.

# Related Work

Besides [1], there have been several studies conducted over the past decade that use large scale internet scans to examine a variety of related phenomena, including entropy, random number generation and other security vulnerabilities in public key infrastructure. In 2008, [8] conducted the self-proclaimed first census of the internet since 1982 using both large-scale internet scans (i.e., censuses) and directed surveys, and found a relatively low level of firewall use across the internet.

In 2011, [9] conducted an analysis of X.509 certificates collected via multiple large-scale HTTPs scan and passive monitoring of TLS/SSL network traffic to examine the robustness of the X.509 key certification process. The results of their study showed that the security of X.509 certificates was lacking as an unreasonably large number of certificates contained errors or were expired, and that many unrelated hosts shared certificates between them. Similarly, in 2011, [10] conducted daily remote scans of over 50,000 SSL/TLS-enabled web servers to examine vulnerabilities in certificates and demonstrated an inherent weakness in the certificate issuance process, in that vulnerable certificates are replaced at a slow pace and CAs continue to issue certificates to weak keys despite knowledge of potential vulnerabilities.

In 2012, [11] analyzed 5.5 million public keys obtained through a variety of opensource methods to assess public key infrastructure and identify vulnerabilities. They found that while most public keys are relatively secure, there was a high rate of public keys that were shared between unrelated entities and “thousands of 1024-bit RSA moduli, including thousands that are contained in still-valid X.509 certificates, offer[ed] no security at all” (11, p. 14). In addition, they found that out of the 4.7 million unique public keys collected, more than 12,000 shared a single large prime factor and posited that this shared prime number pointed to a possible lack of entropy in random number generation, making them highly vulnerable to exploitation.

Finally, in 2014, [12] conducted internet-wide SSH scans and mappings using DNS scans, AS, WHOIS lookups and a geo-IP database, to examine the SSH landscape, including deployment practices and key management. Their findings reproduced the work in [1], in that they found a small number of weak SSH keys but a larger number of duplicate keys, which they deemed ‘strong’ despite the duplication. In addition, they also found that the rate of software updates was slow and old software was prevalent on the internet. Lastly, they concluded, in general, the security (i.e., key lengths) of SSH keys was strong.

In addition to academic studies, several large-scale internet scans have been completed by various external parties. For example, in 2010, [13] released an SSL survey examining domain name registrations and SSL certificates. Through the survey, they collected 867,361 unique certificates, representing what they believed was approximately 25-50% of all commercial certificates on the internet (at the time) and of these certificates. After analyzing the certificates, they concluded that “virtually all deployments have good key size, support good protocols and [possess] strong crypto” (13, p. 37). However, 99 weak keys were recovered and 3,005 certificates with short (i.e., less than 512) key lengths were found, indicating areas for improvement. In the same year, the Electronic Frontier Foundation (EFF) conducted a study [14], scanning and collecting all HTTPs X.509 certificates on the internet to identify security vulnerabilities and trusted hosts (i.e., security companies). They also found through a high number of shared keys, used by multiple CAs as well as many (over 28,000) vulnerable certificates, with minimal entropy (15-17 bits).

# DATA COLLECTION AND ANALYSIS

## Data Processing Method

Two online resources, Censys [15] and Shodan [16], provide access to current public key data. Enterprise access to these databases is provided on a cost-for-service basis, or, for academics, upon approval of an application for access. The original intent of this study was to rely on these resources to obtain both TLS and SSH public key information for analysis and applications were made to both.

However, academic access for this study was not provided for several months. As such, the decision was made to develop original scripts for the collection of TLS and SSH public keys. Due to numerous challenges encountered with data collection, some of which are discussedan original script was developed and used for TLS public key collection, while Shodan resources were leveraged to obtain SSH data.

We have provided some of the high-level details of our work methods here to help researchers with their efforts to reproduce our work in the future. However, to avoid providing potential attackers with a guide for network scanning without detection. While it is likely that most sophisticated attackers already have such software, we have chosen not to include all the details of our work methods and techniques (i.e., no “How-To” guide for ‘script kiddies’). We invite legitimate researchers to contact us, if required.

### TLS Public Key Collection and Challenges

Initially, the resources of Compute Canada [17] were leveraged to scan the entire IPv4 address space and store all certificates presented during either TLS or SSH handshakes, Early collection estimates demonstrated that the use of Nmap scanning and OpenSSL command line interface would take several years. Instead, a script was used to collect the public keys.

In [1], the collection of both TLS and SSH public keys was done individually, and in two separate phases. In the first phase, the IPv4 addresses of online hosts who were listening on the respective port was discovered and recorded. In the second distinct phase, an attempt to connect to and then handshake with, each of the identified hosts was made By this time, only 44.18% of hosts were still listening on the same respective ports.

To improve upon this, the host discovery and certificate collection was changed to occur on both ports in a single pass. For each host, an attempt to connect on port 443 and 22 was made simultaneously using asynchronous concurrent processing. This has the inherent benefit of reducing timeouts and avoiding needing to visit the same IPv4 address multiple times. Furthermore, a single process could asynchronously await multiple hosts and so speed up thecollection of public keys from the IPv4 address space.

The collector script was designed to use a thread-safe queue to store the generated IPv4 addresses Each process of the process pool reads an IPv4 address from that queue, collects the certificate, stores it in an SQLite3 database, and continues until the IPv4 address queue is exhausted. This results in a substantial number of forked processes to obtain the speed required to scan the entire IPv4 address in a realistic timeframe.

#### Resource Consumption

. Unfortunately, with this method, the original script quickly consumed all of the resources of Compute Canada and negatively affected other users (to whom we apologize). To combat this issue, only the the first three octets of the IPv4 addresses were generated, instead, and placed into a thread-safe queue. The processes responsible for the public key collection dequeued the IPv4 addresses and generated all possible 256 IPv4 addresses by filling in the last octet.

This also had the additional benefit of greatly reducing the time the address generation took to complete. In addition, the number of parallel processes was reduced and the SQLite3 database was removed, since the write locking also slowed down the script considerably. Instead, a python logging module was used to write the collected public keys to disk.

#### Pitfalls of Sequential IP Address Collection

While fast and efficient, the script also triggered a European internet service provider’s malicious activity detection algorithm who requested that we desist from further scanning of their servers (reported to us via Compute Canada). Our scan of sequential IP addresses resembled the common behaviour of an infected computer scanning their network for vulnerabilities. Further amendments were required to the collection script to avoid detection and to prevent overloading the servers that were being scanned.

There are several network-level detection mechanisms in place, with reporting and black-listing mechanisms to detect and prevent comprehensive network scans (like ours!) from being executed. We had issues with: Compute Canada, Amazon Web services and even our own local Internet Service Provider. These were obvious in hindsight.

Having 10 parallel process asynchronously connecting to 256 IPv4 addresses at a time caused a large amount of congestion. With 60 parallel processes each collecting from 256 IPv4 addresses, 15,360 TCP connections are required for the port 443 SSL collection alone.

To address this issue, the number of parallel processes was decreased from 60 to 15. To compensate for the loss of parallelism, the peer connection timeout was decreased from 5 seconds down to 2 seconds. Also, a virtual machine was prototyped, with one shared virtual CPU, 2GB of memory, and 10 GB of storage space, to determine the speed at which public keys could be collected. Initial calculations and experiments showed it could take well over a month to finish. To improve the rate of collection, a personal desktop computer, which contained an Intel i7 CPU and 32 GB of Memory, was added to work in parallel.

#### Google Cloud

Having exhausted most of the available resources online and been identified as a ‘threat’ by the threat detection reporting mechanisms of several service providers a decision was made to leverage the resources of Google Cloud. Data collection was disabled on port 22. Using a free[[1]](#footnote-1) account, three five virtual machines were created on Google Cloud at the following locations: Montreal, Iowa, South Carolina, Seoul, and Netherlands. Each virtual machine had two virtual N2 CPUs, 4 GB of memory, and either 10 GB or 20 GB of hard drive space.

Using this method, data collection using the collector script was initialized across all five instances simultaneously, at a rate of 9,600 peers a second, which resulted in a scan of the entire IPv4 address space in 4.5 days and the collection of a total of 38,243,620 TLS public keys. The total cost of collecting the public keys from the entire IPv4 address space was $231.65: CPU: $146.07 (with a $6.11 sustained CPU usage cost reduction) Memory: $18.05 Storage: $0.85 Network: $66.68.

#### Data Compilation

Upon collection of the public key data using the methods above, the data needed to be parsed and stored to support data analysis. The recorded X.509 certificate needed to be further parsed and the data contained within, into an SQLite3 database. Typically, the fingerprint of a X.509 certificate is based on the entire certificate itself. To save space and key comparison time later, a hashed fingerprint version of the key was used.

The parser has a single process that continually reads and buffer the data from the log file into a thread-safe queue. In turn, this queue is then processed by a process pool that places the processed data into another queue. The content of this next queue is then read by a separate process, which continually writes this data to the certificate database.

### SSH Public Key Capture and Issues

Once academic access was granted, Shodan provided all the SSH public keys collected for one month, for the year 2017 as along with full access to daily scan data, which consists of all the data crawled on a specific day. This was contained in JSON format and compressed with Gzip. Even so, the average size of the data for one day was nearly 200GB.

The daily scan data from Shodan appears to be a collection of data from multiple crawlers, each crawling its own set of ports. Furthermore, the crawlers appeared to be going through IPv4 addresses at different rates with no indication of when a scan was started or finished. This begs the question whether the daily scan data across multiple days actually represents a full and complete scan?

To address this problem, Shodan’s command line interface (CLI) was used to parse the daily scan data. However, the output of parsed data for certain days appeared to vary greatly, causing parser failure This bug was traced to a python library called “Click”, unable to write specificUnicode characters to disk and crashing the CLI program. After tracing the problem to an “echo” method in the util.py module in Click’s library, python’s built-in replace was used on the data to correct the issue, when these characters were encountered.

After this, daily data was collected that was representative of a full port scan. The daily data was parsed with Shodan’s CLI and then added it into a SQLite3 database. This database ensured that only new information was being added by having set unique constraints on the IPv4 addresses as well as the fingerprint of the public key. Therefore, when a daily scan reached a point where little new data could be added to the database, it was assumed that a full port scan cycle had been reached.

## Determining Private Key Information from Public Key Data

Two separate methods were used to determine private key information from the public key datasets: FastGCD and a comparison of public keys against default manufacturing key pairs.

### FastGCD

Potential vulnerabilities in the public keys were identified using the FastGCD application developed by [1]. FastGCD uses separate RSA keys to identify keys generated with a common prime number using Bernstein’s algorithm [18]. For this study, a list of distinct moduli was obtained through a SQL query of the public key database. This list was then input into the FastGCD application, which identified vulnerable moduli. Due to the large number of distinct moduli identified (14,764,902), and FastGCD’s requirement to have most of the moduli in memory at a time, the desktop computer used for this calculation with 32 GB of memory continued to crash as it had insufficient available system resources. To rectify this issue, and run FastGCD successfully, a further 64 GB of dedicated memory in the form a Linux page file was required. This was in addition to creation of the intermediate temporary files FastGCD creates for each row of the table requiring many gigabytes of available disk space. In total, FastGCD identified 1,366 vulnerable moduli shared across 13,805 unique TLS hosts and 822 vulnerable moduli shared across 2,579 unique SSH hosts.

### Manufacturer Default Keys

Using the resources of Little Black Box [19] and House of Keys [20], a list manufacturer default key pairs were obtained from the data sets. Using the public keys from these sources and comparing these key lists against the data in our public key database, specific hosts can be associated with specific public keys (for which known private keys exist). Using this method, atotal of 119,862 TLS hosts and 146,033 SSH hosts were identified using House of Keys and 21 TLS hosts using Little Black Box.

## Identifying Device Models

To identify vulnerable device model numbers, each vulnerable modulus present in the list of vulnerable moduli obtained through FastGCD was then queried against the database to create a temporary table, containing all the affected devices. This table was used to group the devices by Organization (the X.509 certificate Organization field the device presented during the handshake). After this, the table was used to produce a list of IPv4 addresses for each Organization grouping.

A connection to each of the IPv4 address was made and any data presented by the peer was scraped and recorded for later analysis. In many cases, a field identifying the model of the affected device was found to have a common descriptor with other affected devices, especially in the case of routers and internet cameras. These descriptors were added to the scraper and the model number could then be automatically extracted for many devices.

Other affected devices also had the model number as the ‘Organization Unit’ field, present in the subject of the X.509 certificate. This data was already present in the SQLite3 database and was obtained by querying and then parsing the result. Finally, by matching the public key to the public key of a known manufacturer default key pair , the specific vendor and model could be identified (i.e. as the host presenting the specifickey in question).

# RESULTS

Using the unique public key collector script, a total of 38.4 million TLS public keys, consisting of 15.1 million unique TLS public keys and 23.2 million repeated TLS public keys, were collected, parsed, and extracted. In addition, using Shodan resources, a total of 14.1 million SSH public keys, consisting of 10.1 million unique SSH public keys and 4 million repeated SSH public keys were collected, parsed, and extracted.

## Identified Vulnerabilities

### Repeated Keys

As outlined above, a significant number of repeated keys were identified across both collected TLS and SSH public keys. Specifically, 60.6% of the TLS public keys identified were repeated and 28.6% of the SSH public keys were repeated. Fig. 1 and 2 below provide a graphical depiction of the top 25 most repeated TLS and SSH public keys, respectively.

Fig. 1: Prevalence of top 25 Most Repeated TLS Keys (per IPv4 address)

As these graphs demonstrate, among the repeated keys, some TLS and SSH public keys were presented a significant number of times. For example, the top 25 TLS public keys presented were repeated over 2.6 million times, with one specific TLS public key repeating 251,391 times alone.

Fig. 2: Prevalence of top 25 Most Repeated SSH Keys (Shodan)

A similar picture was found for SSH public keys, where the top 25 SSH public keys presented were repeated over 1.1 million times, with one specific SSH public key repeating 228,690 times alone. Key re-use, while far less than ideal, might not be a security issue, provided the corresponding private key is not being distributed or configured across multiple devices.

### Accessible Private Keys

Using the two methods outlined in the previous section, the private keys for 242,850 public keys were identified: 134,197 TLS public keys and 108,653 SSH public keys. Using FastGCD, the common prime number being used to create the modulus for some keys was identified and used to obtain 13,888 TLS private keys and 2,072 SSH private keys.

### Vulnerable Device Models

Vulnerable devices were identified across 55 device vendors. It is important to note, however, that a significant number of private keys could not be narrowed down to a specific vendor due to the shared chipset. Some of the weak key generation issues have already been traced back to production process issues for specific vendors, but other investigations are still ongoing.

Fig. 3: Number of TLS Private Keys Obtained by Device Vendor (100 or More)

Fig. 4: Number of SSH Private Keys Obtained by Device Vendor (100 or More)

Fig. 3 and 4 above provide a breakdown of private keys obtained by device vendors, with a minimum of 100 private keys. The data has been anonymized to obscure vendor information. As depicted in Fig. 4, the top five vendors with accessible TLS private keys represented the majority of TLS private keys collected. As shown in Fig. 5, there was a greater variety in the top device vendors for which SSH private keys were obtained than seen with TLS private keys.

# ETHICAL OBLIGATIONS

As the data collection and analysis conducted through this study exposed potential security vulnerabilities across device models and vendors, the decision was made to present these findings to some of the impacted device vendors for further analysis and confirmation. Through secure communication, impacted vendors were provided some of the relevant data and a limited amount of pertinent information on the study methodology to help them quickly identify the issues.

At the time of writing, two of these vendors have confirmed that they will be releasing a firmware update to address the issue. Another indicated that they would be changing their method of certificate handling. Other vendors contacted are still in the process of examining our detailed findings.

# CONCLUSIONS AND RECOMMENDATIONS

We have demonstrated that many both TLS and SSH public keys are shared between hosts. More importantly, there are a significant number (over 300 thousand) of non-random public keys that allow private keys to be easily determined using public sources, making these hosts vulnerable to various threats, including but not limited to man-in-the-middle attacks[[2]](#footnote-2) [21]. Additional analysis.

There are some significant differences in both the timing and quantity of the keys gathered in the original study, compared with our own. Therefore, it is difficult to drawfirm conclusions about whether the problems of shared keys, entropy and vulnerable devices have worsened or not. Numerically, it would seem to be the case. Our results are also very consistent with the body of work in this area [9, 11, 12, 13, 14]. However, we are confident in concluding thatthe state of TLS and SSH public keys and vulnerable devices is still a significant concern.

Despite some of the challenges discussed here, there is a significant amount of analysis that could be done on the public key datasets we have captured for this study. Specifically, we recommend that the specific causes of the lack of key entropy. This work really needs to be done by each affected vendor.

In most cases, the vendors that we informed had well-defined mechanisms for accepting vulnerability reports from a third party, which is encouraging. Most of these vendors acted very quickly (i.e., within hours and days) using secure channels to communicate with us and then began their analyses promptly, escalating the issue to design teams, as appropriate. Unfortunately, this was not the case for all vendors.

Therefore, in publishing this work, we hope to reach all these vendors again and convince them to investigate. Similarly, we hope that any other vendors, who we were unable to identify prior to publication (e.g., vendors who are unidentified in their issued website certificates) can take advantage of our work too. To this end, once again, we extend an offer to provide information and assistance to all legitimate parties with the required information to help them identify their specific problems Once the specific causes of the entropy failures have been determined by each affected vendor, we recommend the addition of those vulnerabilities to the public vulnerability databases that are designed for the purpose of archiving such information.

##### Acknowledgment

D.H. thanks J.F. for her immense support and collaboration in the writing of this paper.

##### References

1. N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, “Mining your Ps and Qs: Detection of widespread weak keys in network devices,” in Proc. 21st USENIX Security Symposium, Bellevue, WA, USA, Aug 2012.
2. “Gartner Identifies Top 10 Strategic IoT Technologies and Trends”. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-11-07-gartner-identifies-top-10-strategic-iot-technologies-and-trends.
3. “What’s new with the Internet of Things? | McKinsey”. [Online]. Available: https://www.mckinsey.com/industries/semiconductors/our-insights/whats-new-with-the-internet-of-things.
4. D. Fernandes, L. Soares, M. Freire, and P. Inácio. “Randomness in virtual machines.” In 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, pp. 282-286. IEEE, 2013.
5. E. Barker, and A. Roginsky. “Transitioning the use of cryptographic algorithms and key lengths”. No. NIST Special Publication (SP) 800-131A Rev. 2 (Draft). National Institute of Standards and Technology, 2018.
6. R. Rivest, A. Shamir, and L. Adleman. “A method for obtaining digital signatures and public-key cryptosystems.” Communications of the ACM 21, no. 2 (1978): 120-126.
7. “Cryptographic Standards and Guidelines | CSRC”. National Institute of Standards and Technology. [Online]. Available: https://csrc.nist.gov/ projects/cryptographic-standards-and-guidelines.
8. J. Heidemann, Y. Pradkin, R. Govindan, and C. Papadopoulos, “Censusand survey of the visible Internet,” in Proc. 8th ACM SIGCOMM InternetMeasurement Conference (IMC), Vouliagmeni, Greece, October 2008.
9. R. Holz, L. Braun, N. Kammenhuber, and G. Carle, “The SSL Land-scape: a thorough analysis of the X.509 PKI using active and passivemeasurements,” in Proc. 11th ACM SIGCOMM Internet MeasurementConference (IMC), Berlin, Germany, Nov 2011.
10. S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage, “When private keys are public – results from the 2008 Debian OpenSSLvulnerability,” inProc. 9th ACM SIGCOMM Internet MeasurementConference (IMC), Chicago, IL, USA, Nov 2009.
11. A. Lenstra, J. Hughes, M. Augier, J. Bos, T. Kleinjung, and C. Wachter ,“Ron was wrong, Whit is right,” Cryptology ePrint Archive, Report2012/064. http://eprint.iacr.org/2012/064.
12. O. Gasser, R. Holz, and G. Carle. “A deeper understanding of SSH: Results from Internet-wide scans.” In 2014 IEEE Network Operations and Management Symposium (NOMS), pp. 1-9. IEEE, 2014.
13. I. Ristic. “Internet SSL survey 2010”. Black Hat USA. http://media.blackhat.com/bh-ad-10/Ristic/BlackHat-AD-2010-Ristic-Qualys-SSL-Survey-HTTP-Rating-Guide-slides.pdf.
14. P. Eckersley, and J. Burns. “An observatory for the SSLiverse”. [Online]. Electronic Frontier Foundation, Available: https://www.eff.org /files/DefconSS Liverse.pdf.
15. Censys, 30-Dec-2020. [Online]. Available: https://censys.io/.
16. “The search engine for the Internet of Things,” Shodan. [Online]. Available: https://www.shodan.io/.
17. Compute Canada - Calcul Canada. [Online]. Available: https://www.computecanada.ca/.
18. D. Bernstein. “.How to find the smooth parts of integers”. .http://cr.yp.to/papers.html#smoothparts.
19. Devttsy0. “Little black box”. GitHub repository. Web site with source code: https://github.com/devttys0/littleblackbox.
20. S. Viehbock. “House of keys”. GitHub repository. Web site with source code: https://github.com/sec-consult/ houseofkeys.
21. F. Callegati, W. Cerroni, and M. Ramilli. “Man-in-the-Middle Attack to the HTTPS Protocol.” IEEE Security & Privacy 7, no. 1 (2009): 78-81.

1. At the time of study, Google was providing a $300 credit for Google Cloud

   Services to all new free accounts. [↑](#footnote-ref-1)
2. As defined in [21], a man-in-the-middle attack is an attack on an HTTPS server, whereby an attacker “replaces a vulnerable certificate authenticating the HTTPS server with a modified one” (p. 78). This type of attack allows the attacker to intercept and divert traffic between the source server and the destination, without the knowledge of either party. [↑](#footnote-ref-2)