# Factoring RSA Keys in the IoT Era

Jonathan Kilgallin
Keyfactor
Independence, OH
jd.kilgallin@keyfactor.com

Ross Vasko
Keyfactor
Independence, OH

***Abstract*—RSA keys are at risk of compromise when using improper random number generation. Many weak keys can efficiently be discovered and subsequently compromised by finding reused prime factors in a large data set. We collect and analyze 75 million RSA certificates from the Internet, and find that 1 in 172 keys share a factor with another. In contrast, only 5 of 100 million certificates found in a sample from Certificate Transparency logs are compromised by the same technique. The discrepancy in rates of compromise is overwhelmingly due to IoT devices exposed to the Internet, which may be subject to design constraints and limited entropy. The widespread susceptibility of these IoT devices poses a potential risk to the public due to their presence in sensitive settings. We conclude that device manufacturers must ensure their devices have access to sufficient entropy and adhere to best practices in cryptography to protect consumers.**

***Index Terms*—PKI, RSA, Internet of Things, Cryptography, Vulnerability Exploit, Network Security**

## I. Introduction

As devices are joined to the Internet and other networks, the potential for network vulnerabilities increases. The recent growth of the Internet of Things (IoT) is driving an expansion in both scope and quantity of network-connected devices. These devices are also now seen in sensitive settings, such as in operating rooms and automobiles. Consumers are also transmitting increasingly sensitive information over the Internet, including financial and personal-health data. The necessity for rigorous security standards is especially important given these elements of the current landscape.

Modern security best practice is to use Public-Key Cryptography to securely transmit data to a remote source – any party can encrypt data with the public portion of the remote source's key, which can then only be decrypted by the remote source's private key. The RSA algorithm has long served as one of the most popular techniques for this process. The security of RSA relies on the inability of another party to determine two randomly-chosen prime numbers from which the RSA public key is derived. If these prime factors are discovered, the private key can be re-derived, and an attacker can impersonate the remote source or decrypt communications that rely on the confidentiality of the private key.

In an attack that has already received a significant amount of attention, these prime factors can be efficiently computed for certain RSA public keys that have been generated with poor randomness, as shown in [1], [2], and [3]. Large numbers of RSA public keys can be mined for common factors between the keys. Each key that is found to have a common factor with another key in the dataset is compromised.

We reexamine this attack and its implications in the current landscape of ubiquitous and sensitive communications between users and devices. The large number of keys that can be acquired from data sets like Certificate Transparency (CT) logs, coupled with the availability of cheap computing resources, means that this attack presents a serious threat if proper precautions are not in place. As part of an ongoing effort to continuously monitor and improve Internet security, we outline an efficient, highly scalable approach to testing the vulnerability of public RSA keys in use. We use the SSL/TLS certificate discovery capability of the Keyfactor platform to build a database of 75 million RSA keys used on the Internet. We then augment this dataset with 100 million certificates available through Certificate Transparency logs. This data is analyzed on a single virtual machine in the Microsoft Azure cloud, using our scalable GCD algorithm for shared factors adapted from [1]. The analysis reveals that at least 435,000 weak certificates – 1 in 172 of the certificates we found on the Internet – are vulnerable to this attack.

## II. Background

### A. The Attack

RSA is used in in the process of encrypting data to send across a network. The server transmits its RSA public key to the client as a part of an SSL or TLS handshake. Part of the RSA public key contains the modulus $n = p * q$, where $p$ and $q$ are two randomly chosen primes of similar size. Primes $p$ and $q$ are kept secret, as knowing these values allows the private key to be calculated. Ensuring that $p$ and $q$ are selected with sufficient randomness is a crucial component of keeping the public key secure. Factoring a large modulus $n$ to obtain $p$ and $q$ is not feasible under normal circumstances. However, if keys are generated with poor randomness, then it becomes a concern that two public keys will share a factor once enough keys are generated. If two RSA moduli $n_1$ and $n_2$ share precisely one prime factor $p$, then computing the Greatest Common Divisor (GCD) of $n_1$ and $n_2$ will reveal the value of $p$ . The GCD computation is significantly easier than straightforward factoring, and can easily be performed in practice. The other factors of $n_1$ and $n_2$ can then trivially be found by the simple calculations $\frac{n_1}{p}$ and $\frac{n_2}{p}$, respectively, compromising both keys. This GCD computation can be scaled to analyze all pairs of keys in sub-quadratic time in the number of keys [1].

Previous research teams have performed large scans on the Internet to collect RSA keys and perform large-scale

978-1-7281-6741-1/19/$31.00 ©2019 IEEE
DOI 10.1109/TPS-ISA48467.2019.00030

GCD computations to demonstrate the implications of this vulnerability. The attack received significant attention in 2012 when researchers were able to break tens of thousands of keys. There was a follow up on the attack in 2016 on a significantly larger data set, with a focus on trends in occurrences of weak keys from various vendors. We focused on three previous publications [1], [2], and [3] in developing our analysis.

### B. Lenstra 2012

The authors of [2] collected approximately 6.2 million digital certificates across the Internet and found that approximately 4.3% of these certificates fully share their RSA modulus with another. The authors clustered the certificates together by modulus to further examine this duplication. Notably, single clusters of sizes 16489, 8366, 6351, and 5055 were found, and 58913 of the clusters have precisely two certificates. Additionally, they found that 12934 of the distinct RSA moduli were able to be factored. The authors found 1200 factors that are shared by more than one modulus. The most widely used individual factors were shared by up to 4627 certificates.

### C. Heninger 2012

In [1], researchers collected 5.8 million unique TLS certificates and 6.2 million unique SSH host keys. Between these two collections, they found 11 million distinct RSA moduli and were able to factor 16,717 distinct public keys. This led to breaking 23,576 (.4%) of their TLS certificates and 1,013 (.02%) of the RSA SSH host keys.

The authors investigated the causes of moduli sharing factors by examining random number generation implementations. A major issue in the flawed random number generation was traced back to a low amount of entropy in the random number pool when the keys were generated. This issue most commonly arises on lightweight devices that do not have much input from which to gain entropy. This analysis found that only two moduli on publicly trusted certificates could be factored, and both certificates had expired. The large majority of the broken keys came from network devices such as routers, modems, or firewalls.

### D. Hastings 2016

The 2012 results of [1] were reexamined in 2016 in [3] with an emphasis on examining how vendors and end-users responded to the vulnerabilities. The authors examine 81 million distinct RSA keys and were able to factor 313,000 keys (.37%). They also examine trends in the number of weak keys from various companies. Since 2012, a significant number of new devices from companies including Huawei, D-Link, and ADTRAN were found to have vulnerable RSA keys. Additionally, it was exceedingly rare for end-users to patch their devices to fix vulnerabilities found during the 2012 study. The authors note the "distressing implications" this has for the IoT era, as the number of devices on a network is rapidly increasing.

### E. 2019

As growth of the Internet and the IoT has continued, network sizes have grown significantly and Internet-connected devices are appearing in new places. Security is a major concern in this landscape because of the increasing amount of data handled by these devices and the introduction of connected devices in sensitive settings such as operating rooms and vehicles. Compromising any device on these critical networks could results in catastrophic failure if proper precautions are not taken. Similarly, consumers are increasingly relying on publicly available web services to handle sensitive data and perform high-impact operations, such as financial transactions, transmission and storage of valuable intellectual property, and applications for credit or other services. The potential for an attacker to obtain this sensitive information or perform fraudulent operations can significantly impact consumers.

Despite the large number of keys broken by this attack previously, it is still unlikely that a key that has been properly generated with a sufficient amount of entropy could be broken with this technique. However, the requirement of sufficient entropy may not always be met when generating keys. In particular, some IoT devices may not have enough entropy available to generate keys without an external source. Major websites can prove vulnerable to this flaw as well, as the public and private keys used with their SSL/TLS certificates are generated by the website owner or administrator, and not by the CA that signs the certificate to make it publicly trusted. This means that the process the site operators used to generate their keys is opaque to the CA, and in general they cannot analyze the submitted public key for poor generation practices. However, there are several reasons that make this issue more prevalent – and potentially more serious – on IoT devices compared to major consumer websites.

*1) IoT is comprised mostly of lightweight devices:* There is nothing inherently insecure about how such a device would generate an RSA key, but [1] found that lightweight devices are primarily at risk of this attack due to their low entropy states. Entropy in a device is required to prevent the random number generation from being predictable. Researchers were able to find deterministic "random" output when removing entropy. Lightweight IoT devices are particularly prone to being in low entropy states due to the lack of input data they might receive, as well as the challenge of incorporating hardware-based random number generation economically. Keys generated by lightweight IoT devices are therefore at risk of not being sufficiently random, increasing the chance that two keys share a factor and allow the key to be broken. The authors of [1] found that most of the keys that broken were from "low-resource" devices. Only two keys on two certificates were publicly trusted, and both of these certificates had expired.

*2) The attack becomes more successful when more prime factors are analyzed:* As the number of discovered certificates grows, the number of certificate pairs to analyze for shared factors increases quadratically. The IoT landscape has drastically increased the number of devices on networks, and this trend is

only expected to continue, although estimates and projections vary considerably.

*3) Devices are in new and more critical environments:* Compromising an RSA key has much more potential to be catastrophic in 2019. An RSA key being compromised now means more than personal or enterprise data being compromised. Critical real-time environments such as operating rooms, automobiles, industrial control devices, and home security systems now operate using RSA keys. Physical property and lives are therefore now at risk with RSA keys being compromised.

*4) IoT devices are more difficult to patch:* Patching many IoT devices across a network can be tedious for a user. There can be many independent devices for a user to consider and manage, often without centralized automation across the devices. In some cases – where original device design did not support patching, or where the device is inaccessible – patching may be impossible. Additionally, due to the long life span of some IoT devices, it is possible that vulnerabilities can be found on live devices that are no longer being actively supported by the manufacturer. This means that a vulnerability to key compromise can be exploitable for much longer, and more difficult to remediate.

*5) Computing and scanning resources are more readily accessible:* Cloud services are readily available for rent to allow an adversary to both acquire and analyze the data efficiently. The only obstacle that an adversary truly faces in this attack is the implementation details. The attack itself has been well studied and understood, and the resources needed to execute it are now readily available at modest cost as well. To illustrate, development and computation resources spent for this study, excluding data acquisition, totaled less than $3000. Furthermore, pre-collected datasets of certificates and keys are available for purchase or even for free download, which can reduce the time and cost for an attacker to perform this computation.

For the above reasons, we reevaluate the implications of this attack in a modern landscape. As in previous works, the goal of this study is to increase awareness of this attack and improve Internet security overall. Given the nature of the research, it is necessary to analyze real-world data. However, as the results compromise keys used to secure real-world communications, we do not provide details on specific broken keys, nor do we retain the broken keys after the analysis

## III. Methodology

### A. RSA Key Collection

The Keyfactor software suite includes the ability to scan an accessible network for endpoints accepting an SSL or TLS connection and serving a certificate. This is applied in enterprise environments as a network monitoring tool – for example, to track expiring certificates. This same scanning technology has been deployed in the cloud and configured to scan the entire public IPv4 address space in an initiative Keyfactor calls "Internet Certificate Inventory" (ICI).

Between April 2015 and November 2017, Keyfactor performed weekly scans by dividing the addressable IPv4 address space between 330 worker machines, giving each worker approximately 11 million addresses. Each worker would attempt a TLS 1.1 connection to each assigned endpoint and collect any certificates returned by endpoints accepting the connection. For endpoints that did not accept a TLS 1.1 connection, the scanning worker would attempt an SSLv2 connection and collect the certificate if it received a response to this protocol. When a worker completed scanning its assigned endpoints, it would transmit all discovered certificates to a central database. Ancillary data, such as endpoints that actively refused a connection, were also collected. Each worker would take approximately 48 hours to complete a scan.

Over the course of 32 months and nearly 140 scans, Keyfactor collected approximately 60 million certificates. These certificates accounted for 45 million unique RSA keys – many keys were re-used by multiple certificates, and a small fraction used other public-key technologies such as ECC. Keyfactor discontinued these scans in November 2017 due to reprioritization of the work. In July 2019, Keyfactor performed one additional scan to support this study. This scan found an additional 13 million certificates, still almost entirely using RSA. To reduce processing requirements, this dataset omitted certificates with RSA moduli that were not broken when considering the original scans.

In addition to these certificates found online, certificates were collected from publicly available CT logs in 2019. Certificate Transparency is an initiative started in 2013 to produce a public record of each certificate issued by publicly-trusted Certificate Authorities participating in the program. To date, there are over a billion certificates contained in at least one Certificate Transparency log. These logs offer a public API to download the certificates, although most APIs limit both the size of the response and the rate at which certificates can be requested. Nevertheless, Keyfactor was able to retrieve a sample of over 100 million RSA certificates with unique moduli in a day across all available logs with a single business-grade commercial PC.

We also included for analysis the lowest 1 million prime numbers. This is a necessary input as some certificates use inappropriately low factors. In fact, some moduli recovered from certificates found online have more than two prime factors! Including these primes allows these keys to be discovered during analysis even if no other certificates used the same factors.

Finally, we considered the 54 "RSA Numbers" [4] – RSA moduli presented as part of a factoring challenge by RSA Laboratories. While the challenge has ended, many of these values have not yet been factored. However, none of these keys were vulnerable to this technique using any other keys in our datasets. This finding is not surprising given the small number of keys, as well as RSA Laboratories' extensive expertise in securely generating RSA keys.

Integrating all of these data sources leads to many duplicated keys. This happens due to the same certificate appearing

in multiple CT logs, certificates in CT logs also being found online by ICI scans, or due to distinct certificates sharing keys. One limitation of this algorithm for breaking RSA keys by identifying shared factors is that, if a key shares both of its factors with at least one other key in the dataset, neither factor can be recovered. Since any key that occurs more than once will share both factors with each other instance of the same key, it is necessary to eliminate any duplicates. This was achievable with a simple script-processing job; we list the hexadecimal value of each modulus line-by-line in a text file, sort these moduli lexically, and then eliminate adjacent duplicates. After removing duplicates, these sources resulted in a total of 158,641,969 unique RSA keys to analyze, occupying 81 GB in this hexadecimal text representation.

### *B. Find Shared Factors*

The core of the analysis is to perform a large-scale Greatest Common Divisor (GCD) computation among the moduli. There is no known polynomial-time algorithm for factoring integers in general, and it is suspected that one does not exist for classical computers. However, computing the GCD between two numbers can easily be done in polynomial time. Suppose there exist two RSA public keys that both have a sufficient number of bits $n_1 = p * q$ and $n_2 = p * r$. If an adversary only knows $n_1$ or only knows $n_2$, it would not be feasible for the adversary to factor the known value given current techniques. However, if an adversary knows both $n_1$ and $n_2$, $GCD(n_1, n_2)$ can be computed quickly. This consequently compromises both keys because the other prime factor can be trivially found by calculating $\frac{n_1}{p}$ and $\frac{n_2}{p}$.

Given RSA public key moduli $n_1, n_2, \cdots, n_m$ where $n_i = p_i * q_i$, if some keys have been generated with poor randomness, then with large enough $m$, it is reasonable to guess that there exist distinct $i$ and $j$ such that $GCD(n_i, n_j) \neq 1$. Naively, one could attempt to break all possible RSA keys by evaluating $GCD(n_i, n_j)$ for all distinct $i$ and $j$. This requires $O(m^2)$ GCD computations and is too inefficient to complete in a reasonable amount of time. Instead, we can perform this much more efficiently using a product tree, a remainder tree, and the following equivalency, proven in [5]:

$$GCD(n_i, n_1 * \cdots * n_{i-1} * n_{i+1} \cdots * n_m) = GCD(n_i, \frac{n_1 * n_2 * \cdots * n_m \bmod n_{i^2}}{n_i}) \quad (1)$$

The second argument for the GCD calculation can be efficiently computed for each $n_i$ using the product tree and remainder tree. This method was described for this application in [1]. Noting that all values in our dataset fall between 512 and 8192 bits, and that there is little motivation for larger keysizes to be used by industry, we treat the size of each input as a constant. The runtime of fully computing the product tree and remainder tree is then given in [1] as:

$$O(m * log^2(m) * log(log(m))) \quad (2)$$

Once we have our set of RSA moduli $n_1, n_2, \ldots, n_m$, we proceed by computing the product tree and remainder tree. At the completion of the remainder tree, we have $n_i$ paired with $k_i = (\prod_{i=1}^{m} n_i) mod n_i^2$. Using the equivalency in [5], we divide $k_i$ by $n_i$ and compute $GCD(n_i, \frac{k_i}{n_i})$ for all $i$. In the case that the GCD is found to be 1, the modulus did not share either factor with any other modulus in the set, and so the public key could not be broken using this technique and dataset. If the GCD is not 1 and not $n_i$, then we have a shared factor and we can break the key.

If the GCD is found to be $n_i$ itself, this means that $p_i$ and $q_i$ both existed in other keys. Since the moduli are distinct, we know there is a key with $p_i$ but not $q_i$, and one with $q_i$ but not $p_i$. We separately record these keys to break at a later step. We first attempt to break these keys by dividing all known factors acquired from the result of the GCD algorithm described above. Almost all remaining keys are broken with this approach. It is possible, however, to generate keys with shared factors that cannot be broken this way – for example, if $n_1 = p * q, n_2 = p * r$, and $n_3 = q * r$, with $p, q$, and $r$ not found in any other keys, they will not be broken with these steps. The small number of remaining keys that are not divisible by factors known at this point are then broken through straightforward pairwise GCD calculations.

For each broken key, we output the broken modulus along with its two factors in one csv file, and simultaneously output a PKCS1 file containing the complete re-derived private key – after factoring the public modulus into its two factors $p$ and $q$, the other computations to derive all values needed for the private key are trivial. We were able to correlate these factored moduli back to the original certificate record using a simple SQL query against the ICI database using an index on the key's modulus. With the PKCS1 private key and the original certificate, we were able to validate our results by verifying cryptographic signatures using Microsoft Crypto Shell Extensions, Microsoft's certutil program, and OpenSSL.

### *C. Our implementation details*

We implement the GCD computation in a C program using the GNU MultiPrecision (GMP) library for arithmetic operations. The algorithm is implemented in parallel and runs across 32 cores on an Ubuntu virtual machine in Microsoft Azure with 432GB of RAM and 1.164TB of premium Solid State Drive swap space.

We encountered significant memory limitation with our algorithm running with 32 cores on such a large data set of RSA keys. Each level of the product and remainder tree requires about the same amount of memory, as the product of two numbers requires approximately the same number of bits as the sum of the bits in the input numbers. Since the $m$ keys require $log_2(m)$ levels, and the large dataset leads to considerable memory usage at each level, the memory requirements are significant. Multiple mitigations to this have previously been considered, notably including distributed computation to more easily scale available resources [3]. Part of our goal, however, was to determine if a large dataset could effectively

be analyzed on a single virtual machine in a modern cloud service. This simplifies the architecture and thus decreases the cost and complexity to implement and perform the analysis, increasing the risk that this vulnerability can be effectively exploited. Here, we considered several different optimizations to allow this analysis on a single machine and implemented three that allowed the computation to fit in available memory.

First, we note that to compute level $i$ of the product tree, the only input is level $i$-1. This means that lower levels can be discarded from memory while higher levels are computed. Then, on the remainder tree, level $i$ only depends on level $i$ +1, which means that higher levels can be freed and will never be needed again. Since level $i$ will have been taken out of memory, it is necessary to reconstruct that portion of the product tree in order to continue the remainder tree computation. However, the product tree is significantly faster to compute, so the lower tree can be reconstructed quickly. Additionally, for any given node in the remainder tree, continued processing only requires nodes directly below that node. This means that the tree can be divided horizontally into an arbitrary number of subtrees, and the number of subtrees considered at one time can be reduced to accommodate available memory. As computation proceeds down the remainder tree and higher rows are freed, more subtrees can be loaded back into memory at once.

Second, we note that in the remainder tree, a given node has two children and four grandchildren. However, the remainder value for the four grandchildren can be computed directly against the grandparent node without ever using the intermediate product or remainder values. To simplify the implementation, we shortcut this process by using four children per node and performing the computation as before without skipping rows. This also further reduces peak memory requirements, as when skipping rows, the intermediate product nodes must still be computed and retained temporarily.

Finally, we note that the highest memory usage is in the first few levels of the remainder tree, but we can reduce memory usage by temporarily reducing the number of active threads, regardless of how many nodes could theoretically be processed simultaneously. A related runtime optimization we considered but did not implement includes utilizing the idle threads for lighter-weight computations, such as computing the squares of the node values for lower levels of the remainder tree. Further optimizations to the GMP library usage, or even using another integer library that provides parallel multiplication and modular arithmetic, could improve performance even more. However, the optimizations we implemented were sufficient to run the computation in available memory. While we conclude that this can be effectively done on a single machine, it is likely that at larger scales, computation would best be performed with a distributed implementation, such as developed in [3].

## IV. Results

When analyzing the 45 million moduli from our 2015-2017 Internet scans, along with the first million low primes, we were able to break 192,709 keys, corresponding to 344,055 distinct certificates in the original dataset, or 0.56%. Twelve certificates failed signature validation when correlating the broken key with the original certificate. These appear to represent minor variations of legitimate certificates that occur in our dataset but were not compromised. The variants all produced an invalid public key, in that it is not a product of two primes, which causes the signature check to fail. These certificates were all CA certificates obtained by multiple ICI scans, so we attribute this to file corruption in the certificate chain on the hosting server.

When incorporating the results of the 2019 ICI scan and the 100 million certificates from CT logs, we were able to compromise 249,553 distinct keys corresponding to 435,694 certificates. Of these results, only five certificates are publicly-trusted certificates found in CT logs, including one domain with two compromised certificates, issued by different CAs. As of this writing, none of these certificates remain in use online. The remainder of the certificates are self-signed, privately-rooted, or device certificates. Excluding the CT logs, these numbers indicate that around 0.58% of certificates online during the scanning period are vulnerable to this attack. Statistical summaries are given in Table 1.

It is interesting to note that some of the broken moduli are used by considerably more than one certificate, with the average number of certificates per key being 1.75. The most commonly occurring modulus value was used by 1737 certificates, and 19 moduli were used by more than 300 certificates. The usage statistics for individual factors are even more concentrated, and Table 2 gives the top ten most used factors. These ten factors together, while less than .01% of the factors across all keys, account for more than 6% of the broken certificates. These factors have been redacted to prevent abuse while giving manufacturers the opportunity to recognize if they've generated a common weak factor; the first and last 8 bytes are provided to facilitate this.

Of these top factors, eight appear to represent factors of an appropriate length for a 1024-bit RSA key and two appear to represent factors of a 512-bit RSA key. Across all reused factors in a typical range for each common key size, the distribution is given in Table 3. In addition, one 2204-bit factor was shared by two keys, along with 16 factors of 13 bits or less found by including the low primes.

As with previous results in this field, we are able to determine some information about the usage of the broken certificate by looking at its subject. Of these certificates, for example, 217,988 – almost exactly 50% of compromised certificates – contain the name of a large network

TABLE I
STATISTICS ON BROKEN KEYS

| **Source** | **RSA certificates** | **Unique moduli** | **Keys broken** |
|---|---|---|---|
| 2015-2017 ICI | 61,721,960 | 44,571,776 | 192,709 |
| 2019 ICI | 13,186,479 | 13,070,139 | 56,839 |
| CT logs | 100M | 100M | 5 |
| RSA numbers | N/A | 54 | 0 |

TABLE II
MOST COMMONLY FOUND FACTORS

| Factor | Count |
|---|---|
| C3B1F6B93F3C0E1E...A18BA0432F4A155B | 8197 |
| C9C7AE3ADF60EBA0…167FA6D24BCD8AB1 | 4913 |
| C680429BF3D695D3...7D3954EA4C557437 | 4614 |
| E8073ED3C66C329A...B8F8EECA5BFFDC2B | 1739 |
| F43088641FDA33FE...A47882D6EB130419 | 1737 |
| D82C3689990CA75B...2E4C905A89335F47 | 1670 |
| DA2DB0BF33F6A3CF...EE1C5DE1E890237D | 1431 |
| F9C1BD6DD6401EF0...13A4B74FE74D894B | 1335 |
| F8DDB61D859BC7A5...C8EA81105FCD490F | 1318 |
| ED405D59FE0D419A...EA234734E1A4CB91 | 1316 |

TABLE III
KEY LENGTH DISTRIBUTIONS

| Factor length (bits) | Presumed key length | Count |
|---|---|---|
| 256-257 | 512 | 29254 |
| 512-514 | 1024 | 213315 |
| 1021-1024 | 2048 | 8744 |
| 2033-2048 | 4096 | 14 |
| 4084 | 8192 | 1 |

equipment manufacturer previously notified as a result of [1] and [3]. This includes 66,939 certificates discovered in the 2019 ICI scan, indicating that the manufacturer and/or consumers of their products continue to have issues with security in the field. In fact, many of these devices used the same key. Unsurprisingly for reasons already discussed, at least 12 of the 20 most common certificate subjects in our dataset appear to represent consumer IoT devices. In some cases, we were able to determine the exact model of device represented. Other common subjects include very simple IP addresses like "CN=192.168.1.1" (25,075 certificates) and subjects that appear to have been taken from examples without modification, such as a "E=HOST@ANYCOMPANY.COM, CN=NOHOSTNAME.NODOMAINNAME, O=ANY COMPANY, L=ANY LOCALITY, ST=ANY STATE, C=US" (242 certificates). As we have not been able to identify or notify all manufacturers, we do not provide further information about specific subjects.

## V. Conclusion

With modest resources, we were able to obtain hundreds of millions of RSA keys used to protect real-world traffic on the Internet. Using a single cloud-hosted virtual machine and a well-studied algorithm, over 1 in 200 certificates using these keys can be compromised in a matter of days. This is concerning, as a party with a re-derived private key for an SSL/TLS server certificate can impersonate that entity, and network clients attempting to connect to that endpoint cannot distinguish the attacker from the legitimate holder of the certificate. In today's landscape where these clients may represent automobiles, medical implants, or other critical devices, the impersonated service may cause the device to malfunction and cause physical harm.

While the particular attack described here is specific to RSA keys, other encryption schemes such as Eliptic-Curve Cryptography (ECC) do still rely on secure random number generation, and other attacks could exploit a lack of entropy in key generation to compromise an ECC key. We consider only one of many potential threats that can cause a key that appears secure – or that was once secure – to be unexpectedly compromised. These concerning findings highlight the need for device manufacturers, website and network administrators, and the public at large to consider security, and especially secure random number generation, as a paramount requirement of any connected system. Systems should attempt to use security best practices from inception, and in any case must have the ability to securely update both software and cryptography to protect against risks like this.

## References


[1] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining your Ps and Qs: detection of widespread weak keys in network device," Proceedings of the 21st USENIX Security Symposium, August 2012.

[2] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, "Ron was wrong, Whit is right", 32nd International Cryptology Conference, August 2012.

[3] M. Hastings, J. Fried, and N. Heninger, "Weak keys remain widespread in network devices," Proceedings of the 2016 Internet Measurement Conference (IMC '16), ACM, New York, NY, USA, 49-63.

[4] RSA Laboratories, "RSA Numbers", http://www.ontko.com/pub/rayo/primes/rsa_fact.html.

[5] B. Cloostermans, "Quasi-linear GCD computation and factoring RSA moduli", August 2012. https://pdfs.semanticscholar.org/4124/edde77caeecbbee378ccdd12ffbaa2e9f322.pdf.