Detecting Weak Keys in Manufacturing Certificates: A Case Study

Andrew Chi
andrchi@cisco.com
Cisco Systems
San Jose, CA, USA

Brandon Enright
brenrigh@cisco.com
Cisco Systems
San Jose, CA, USA

David McGrew
mcgrew@cisco.com
Cisco Systems
San Jose, CA, USA

ABSTRACT

Weak entropy is an industry-wide challenge for network device vendors. We conducted a large scale analysis of RSA keys in about 226 million device certificates from one vendor, covering products that were manufactured over a 12-year time period. By focusing on specific data features of the manufacturing certificates, we tested for common keys and common factors across distinct devices. The scale of our analysis enabled the detection of entropy failures that manifested in the RSA keys of millions of devices. The affected devices included several products not implicated in any prior studies, resulting in the discovery of three new vulnerabilities in actively supported products. The entropy failures were complex, resulting from both low initial entropy and the faulty composition of manufacturing processes. Most affected product families were lower-margin devices past their end-of-support date; higher-end products that used a vendor-sanctioned hardware entropy source did not exhibit these weaknesses. However, our findings warrant more proactive and systematic entropy testing by device vendors.

CCS CONCEPTS

• **Security and privacy → Key management**; *Cryptanalysis and other attacks*; **Public key (asymmetric) techniques**.

KEYWORDS

device certificates, entropy, RSA, key generation

ACM Reference Format:
Andrew Chi, Brandon Enright, and David McGrew. 2023. Detecting Weak Keys in Manufacturing Certificates: A Case Study. In *Annual Computer Security Applications Conference (ACSAC '23), December 04–08, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3627106.3627120

1 INTRODUCTION

Cryptography relies on the unpredictability of keys, and secure key generation requires a good entropy source. Despite this widespread need, the implementation and testing of entropy sources remains a challenge for the industry. The widespread prevalence of weak public keys on Internet devices was demonstrated twice over the last decade [10, 11], and persists today, as we show in Section 4.1.2. Software entropy sources are especially challenged when tapped immediately after bootup, when their entropy pool is effectively empty. This Low Initial Entropy (LIE) problem has plagued network devices, many of which generate a public-private keypair during their first run, either during the manufacturing process, or on the customer's network.

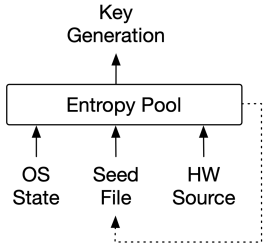

Figure 1: Entropy can be gathered from specialized hardware, or through operating system software. Some devices store entropy across reboots.

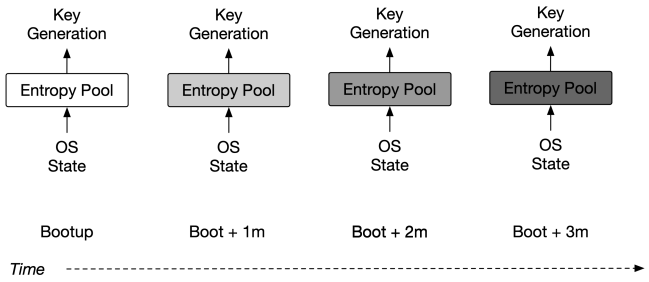

Figure 2: Software entropy sources accumulate unpredictability over time, and its outputs may be weak for some period after startup.

Many operating systems harvest entropy from their environment by gathering together many slightly unpredictable values and accumulating entropy over time (Figure 2). However, it is difficult to verify the soundness of such designs, and the amount of entropy they produce can depend on factors like their operating environment and hardware and software configuration. Some devices store entropy across reboots in a 'seed' file, so that the pool is not empty after a restart (Figure 1). However, these devices still have low entropy after their first bootup.

Determining whether an entropy source is susceptible to low initial entropy (LIE) can be challenging. Fundamentally, it is impossible to make a statistical assessment based on a single output. Restated intuitively, there is no such thing as a random number—there are only methods to randomly produce numbers. Therefore,

ACSAC '23, December 04–08, 2023, Austin, TX, USA

ACM ISBN 979-8-4007-0886-2/23/12…$15.00
https://doi.org/10.1145/3627106.3627120

multiple samples are always necessary for entropy testing. Unfortunately, obtaining multiple samples is necessary but not sufficient. Repeatedly sampling a *single* device a large number of times *will not* detect LIE, as a typical pseudorandom generator (PRG) will take an initial seed value and create a sequence of numbers that do not collide,[1] even if no new entropy is ever mixed into the PRG state. Sampling *many* devices at boot time *will* detect LIE: devices with the same initial state and the same nondeterministic inputs will generate the same sequence of random numbers. This is detectable by generating a small sample of numbers from each of a large collection of devices, i.e., population testing.

Given a population of devices, what test should be applied? Testing devices for low initial entropy could conceivably be performed at many different system layers: hardware or software component unit tests, operating system regression tests, or run time checks. This work focuses on an output that represents an end-to-end test: device X.509 certificates for use with TLS. This choice is important. While unit tests can check for good entropy at a component level, good components can still be composed into a faulty system. Device certificates—specifically the public keys—therefore represent an end-to-end test that samples entropy in the form of key generation output.

How does weak entropy appear in device certificates? We examined two catastrophic failure modes: (1) common factors, where two or more RSA keys, generated by distinct devices, share a common factor, and (2) common keys, where two or more keys, generated by distinct devices, are identical. To test for common factors in a large population of devices, we used the batch greatest common divisor (BGCD) method developed by Bernstein [3] and implemented by Heninger et al. [11]. In order to accommodate the scale of our dataset, we reimplemented the GCD method to scale beyond the limitations of the GNU Multiple Precision Arithmetic Library (GMP). We also adapted the GCD method to optimize for cloud computing cost. To test for common keys, we grouped certificates with the same public key. We then developed vendor-specific code that used other certificate fields to determine whether two certificates with the same public key came from the same device (permissible) or from two distinct devices (entropy failure). In addition, for products with common factors or common keys, we estimated the number of bits of entropy actually exhibited in the typical key space. This estimate could be useful both for debugging (identifying the faulty component), as well as estimating the cost to an attacker whose goal is to purchase enough devices to observe collisions.

We applied the common-factor and common-key tests to two large datasets. The first dataset consisted of manufacturing certificates. Manufacturing certificates are installed after a device is assembled but before it is shipped to a customer. These certificates can be readily obtained from a centralized point, as it is standard practice for a device vendor's Certification Authority (CA) to keep records of all certificates that it has ever issued. In the case studied, the dataset consisted of about 226 million manufacturing certificates issued from 2008-2020 by a vendor certification authority (CA), namely Cisco Cryptographic Services . The second dataset consisted of 37.7 million TLS certificates obtained from internet scans during Spring 2021, performed by Rapid7's Project Sonar. Within this dataset, we especially focused on device-issued certificates, e.g., the certificate generated when a user enables an administrative interface on the device and chooses to protect that interface using TLS.

The majority of manufacturing certificates were not affected, including all devices that used a hardware entropy source that was vendor-designated for its higher end products. However, we did discover previously unknown weak keys in over 4 million device certificates issued over the span of 2008 - 2020. These weak keys affected several product families, including Aironet-based WLAN, IP telephony, and Linksys. Weak keys started appearing in 2008, hit a peak of 250,000 per month in 2012, and have tapered off, with few appearing in 2020. Most affected products were past End-of-Life (EOL), but the CP-6901 had this problem, and was not yet EOL, as did the Cisco 2504 Wireless Controller, which was not yet End-of-Support (EOS). We also found weak keys in self-signed certificates of Cisco Small Business Routers (RV220W, RV130W) obtained from internet scans. For the devices that were not End-of-Life, we worked closely with the vendor's Product Security Incident Response Team (PSIRT) to publish one new CVE (CVE-2022-20817) and two new release note enclosures advising of product vulnerabilities.

This case study shows that the low initial entropy problem still affects devices currently being manufactured, and can result in catastrophic entropy failure in RSA public keys. Fortunately, in the vendor studied, the prevalence of the problem has decreased over the years, and is restricted to a small subset of the vendor's products. Nevertheless, based on the results in this study, we recommend population-wide testing of entropy in manufactured devices, after the device has been assembled, but before it is shipped to the customer. Such testing can potentially detect entropy failures before products are shipped to a large number of users.

2 CONTRIBUTIONS

This study is the largest-scale analysis to date of both manufacturing X.509 certificates, as well as common (duplicate) keys in those X.509 certificates. Investigating common keys has been challenging to do in previous studies due to the difficulty of distinguishing whether two certificates with the same key were issued to the same device (legitimate) or different devices (entropy failure). By working closely with one vendor's certification authority, PSIRT, and product teams, we were able to differentiate between devices and investigate entropy failures. Detection of common keys is applicable to ECC and atomic RSA key generation, not just non-atomic RSA key generation, and therefore enables our system to detect flaws in a larger class of implementations. In addition, we apply a mathematical analysis to calculate the poorness of the entropy source that fed key generation. This enables us to estimate how many keys are needed to test the devices, and in the case of low entropy, gives debugging clues as to which entropy sources may have failed. Finally, we apply batch GCD to a substantially larger set of RSA keys than previous works, by parallelizing the computation in a way that scales well to cloud computation and does not require modification of the GMP library.

[1] More precisely, given a family of statistical tests, a PRG maps a random seed to a longer string of bits that no statistical test in the family can distinguish from the uniform distribution. Most random number generators are built from a PRG.

3 RELATED WORK

We build on a line of prior works. Bernstein [3] developed a batch factorization algorithm that quickly computes all-pairs GCD among n integers by using a product tree, remainder tree, n division operations, and n GCD operations. In 2012, Heninger et al. [11] applied Bernstein's algorithm to 11.2 million RSA keys obtained through internet scans for TLS certificates, discovering a surprisingly large number of both repeated keys and factorable keys. (Due to the nature of the dataset, repeated keys could not always be clearly mapped to distinct devices.) In 2016, Hastings et al. [10] repeated the study on 81 million RSA keys found through subsequent scans, and found that weaknesses remained. They were the first to parallelize the batch GCD computation into subsets, and made modifications to GMP to accommodate the large integer products (the authors generously shared their GMP patch code). In 2019, Kilgallin and Vasko [12] analyzed 159 million RSA keys from internet scans and Certificate Transparency (CT) logs for common factors but not common keys. They found that broken moduli almost entirely came from the internet scans, and they attributed over half of broken moduli to Internet of Things (IoT) devices. Our work builds on this line of research, but focuses on manufacturing certificates that are not usually accessible via direct internet scans; we worked closely with the vendor to identify distinct devices and to investigate the root cause of entropy failures. We also wrote and open-sourced a batch GCD implementation that can be parallelized to handle over 226 million certificates (using cloud compute resources). Our implementation differs from Hastings et al. in that it further decouples the subset computations, and does not require patching GMP, making it more easily reusable and maintainable in the future. Moreover, our work not only detects factorable RSA keys, but also common keys; to our knowledge, no recent work has performed a thorough investigation of common keys.

An attacker wishing to exploit weak RSA keys may need to determine the library that generated those RSA keys. Svenda et al. [20] developed methods for origin attribution of a large sample of RSA keys. Branca et al. [6] were able to achieve high attribution accuracy in some cases even with a single RSA key. In cases where the private key is known or recoverable (e.g., via batch GCD), there can be further attacks that model the prime-generation procedure. Bernstein et al. [4] factored RSA keys in smart cards initially using batch GCD, and after noticing visible patterns of non-randomness in the recovered primes, applied a Coppersmith-type partial-key-recovery attack to recover additional private keys.

Our work in estimating the size of the typical set of keys (and thus the effort required of an attacker) has parallels to estimating animal populations using a mark-recapture methodology [15]. The problem statement that we chose is known as the Inverse Coupon Collector's Problem, and has been studied by Dawkins [9] as well as Langford [13]. Our formulation is of course equivalent, but can be implemented cleanly in a few lines of Python (using SciPy [22]) without requiring optimization of an objective function that is a summation where the number of terms depends on the input variable over which we are optimizing.

While our batch GCD work targets RSA keys, elliptic curve cryptography (ECC) is also widely deployed and can also be vulnerable to both accidental and intentional weaknesses in random number generation. Bos et al. [5] studied several practical deployments of ECC and noted weaknesses due to apparent entropy failures; for example, poor randomness used in ECDSA signature generation could compromise the long-term signing key. Checkoway et al. [7] analyzed the Juniper Dual EC incident, which involved a pseudorandom number generator whose outputs could be readily predicted by an adversary capable of choosing the elliptic curve parameters.

3.1 FIPS-140

Historically, the entropy tests in the U.S. Government's FIPS-140 certification testing program [16] would not detect Low Initial Entropy, but recent changes to that program add a "restart" test that aims to close that gap [18]. See Section 7.18 of the NIST CMVP Implementation Guidance [17], and Section 3.14 of NIST Special Publication (SP) 800-90B [21]. At least some affected products have been through FIPS-140 evaluations (Such as the CT2504, for instance), which underscores the incompleteness of those historical evaluations.

3.2 Pseudorandom Generator Cloning

Weak entropy has an an important parallel in the issue of pseudorandom generator cloning, which can undermine cryptographic security in virtualized environments in ways that escape most security controls. A cloned snapshot of a running VM, or an application `fork()`, causes misuse of pseudorandom generator state that can render TLS and other cryptographic sessions insecure (Fig. 3). These problems can be detected using a batch methodology similar to the common key test. Preliminary work in this area found several issues on a corporate network [14]. It should be possible to extend the network metadata capture tool to report TLS random nonces, to enable collection from operational sessions at scale. It may also be worth investigating signatures and public keys.

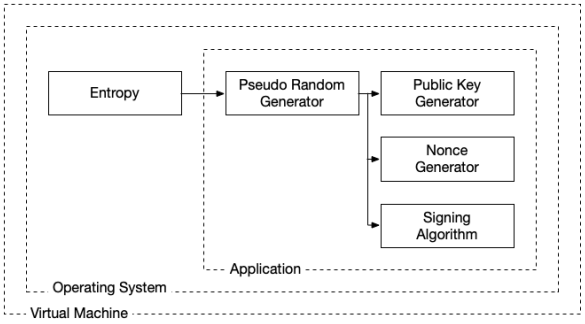

Figure 3: A running VM snapshot, or application `fork()`, inadvertently duplicates pseudorandom generator state, which can lead to duplicate keys and nonces that undermine security.

4 METHODS

To test for LIE, it is *not* sufficient to test many keys that are generated successively during a single run of a single device, which has been the dominant methodology for entropy testing[2]. However,

[2]FIPS-140 accepted the single-run methodology up until 2021; see Section 3.1.

testing many keys generated right after bootup *can* detect LIE. To check for this problem in Cisco products, we tested the subject keys in manufacturing certificates (Section 4.1.1) obtained from the Cryptographic Services team, and a set of device-issued certificates obtained from internet scans (Section 4.1.2). Manufacturing certificates are issued by the Cisco Certificate Authority (CA), whereas device-issued certificates are self-signed, but in both cases, the device may generate its own key, and may do so soon after booting up, possibly for the first time.

We used two tests that processed all of the certificates in bulk:

- the **batch greatest common divisor test** (BGCD) identifies RSA public keys that share a **common factor**, and
- the **common key test** identifies devices that share a **common key**.

A common factor or a common key, in certificates issued to different devices, indicates a significant entropy failure[3]. When applying these tests to certificates, without any other ground truth data about which device generated which key, the BGCD test is important because a common factor is unambiguous evidence of weak entropy that needs no further analysis.

The tremendous size of certificate data set posed a challenge: it was several times larger than the BGCD data sets previously studied in the literature. There was no available implementation of BGCD that could scale to the size of our data set. We overcame this obstacle by developing a scalable and performant BGCD implementation[4] and certificate parser[5], and by running these tools on a Google Cloud Platform (GCP) machine with 1TB of RAM, using a data-sharding approach. See Section 4.3 for further details.

4.1 Datasets

4.1.1 Manufacturing Certificates. Manufacturing certificates follow the IEEE Secure Device Identity standard [1], a profile of PKIX/X509 [8], and are issued by Cisco Cryptographic Services after a device is assembled but before it it shipped to a customer. Within Cisco, they are termed Secure Unique Device Identifiers (SUDI). For some products, the public key in a manufacturing certificate is generated by the device associated with it, while other products have keys that are generated by an external Hardware Security Module (HSM).

We analyzed 226,131,301 certificates issued by Cisco's Cryptographic Services team up through 2020. The majority of these certificates had 2048-bit RSA subject keys. Each SUDI certificate usually contained a distinct Product Identifier (PID) and a distinct Serial Number (SN) in its subject field. We used the PID to identify products, and we used the SN to identify distinct devices (Section 4.4).

4.1.2 Device-Issued Certificates. Many products do not have manufacturing certificates and instead generate a self-issued certificate on the customer's network. To extend the coverage of our testing to include some of these products, we downloaded a set of certificates that were obtained from internet scans during Spring 2021. We used Rapid7's Project Sonar, which provided access to X.509 certificates observed when scanning HTTPS endpoints on the public internet. It records certificates *only* when they were not seen in previous scans, so this dataset should be interpreted as TLS certificates that were newly deployed or newly reachable in Spring 2021, specifically between February 6, 2021 and May 5, 2021. We identified the certificates generated by Cisco products by the presence of the string `Cisco` in the certificate issuer field.

The Spring 2021 data set includes 37.7 million distinct certificates, only 0.1% of which are from Cisco devices, as detailed later in Table 3. It includes over 150 distinct Cisco PIDs (Table 1), associated with Small Business Routers, IP Phones, Unified Computing Systems (UCS), and WLAN access points.

Certificates	Cisco Product ID (PID)
3876	RV042
3254	RV320
2274	RV042G
1809	RV325
519	RV130W-A-K9-NA
397	RV130W-E-K9-G5
393	RV110W-A
380	RV130-K9-NA
341	RV082
284	RV130-K9-G5
260	RV110W-E
180	CP-8841
152	RV180W
149	RV215W-E
142	RV215W-A
134	RV180
115	UCSC-C220-M5SX
110	RV220W
94	RV016
83	AP1G5
74	UCSC-C220-M4S
61	RV120W
58	CP-8851
55	CP-8861
48	CP-7841
36	RV130-WB-K9-NA
34	CP-8865
33	UCSC-C220-M3S
31	CP-7821
29	CP-8811

Table 1: The most common Cisco product IDs (PIDs) present in the Spring 2021 Sonar SSL data set. The complete list of Cisco products observed includes over 150 distinct PIDs.

4.2 Detection Methodology

There are two main ways that weak entropy shows up in certificates:

Common factors - Two or more RSA keys, generated by distinct devices, share a common factor. This only occurs for RSA keys, and it is is incontrovertible evidence of weak entropy.

[3]Section 4.5 describes how to exploit these flaws.
[4]The C++17 tool `batch_gcd`; see also its documentation, which can be found at `https://github.com/cisco/mercury/blob/main/doc/batch-gcd.md`.
[5]The C++17 tool `cert_analyze`.

```
void generate_rsa_private_key_nonatomic() {
   P = find_prime(get_pseudorandom(get_entropy()));
   // the entropy pool may be changed here
   Q = find_prime(get_pseudorandom(get_entropy()));
 }

void generate_rsa_private_key_atomic() {
   X = get_entropy();
   P = find_prime(get_pseudorandom(X));
   X = advance_pseudorandom(X);
   Q = find_prime(get_pseudorandom(X));
 }
```

Figure 4: Pseudocode illustrating RSA private key generation algorithms that are non-atomic (top) and atomic (bottom) regarding entropy use. When used with a weak entropy source, non-atomic RSA key generation can exhibit either common factors or common keys, while atomic RSA key generation can exhibit only common keys. The common factors issue is specific to RSA, and is inapplicable to Diffie-Hellman key establishment, including its elliptic curve variants, and other public key cryptosystems.

Common keys - Two or more keys, generated by distinct devices, are identical. This can occur for any cryptosystem, but it is harder to detect through the automated batch analysis of certificates, because it is acceptable for two certificates from the *same* device to contain the same key. The distinctness of the devices can only be determined by analyzing subject or subject alternative name fields.

Some implementations of the RSA key generation algorithm will never exhibit common factors and will only exhibit common keys in weak entropy situations. Common factors occur when the entropy pool changes in between the generation of the first and second prime factor, and this can only happen if the algorithm is not atomic with respect to entropy use, as illustrated in Figure 4.

4.3 Batch GCD

The Batch Greatest Common Divisor (GCD) algorithm is a clever way to identify common factors in a (potentially very large) number of RSA public keys [11]. It is computationally intensive, requiring a random access memory large enough to hold a representation of the product of all of the integers in the batch. If directly applied to the large datasets of RSA keys in this study, the product would exceed the memory capacity of any single cloud instance, as well as the maximum integer limit of the GNU Multiple Precision Arithmetic Library (GMP). We instead split the RSA moduli into subsets and ran batch GCD on all pairs of subsets, similar to Hastings et al. [10]. While appearing to introduce quadratic computational complexity, surprisingly, this actually optimized for both speed as well as monetary cost. Speed was increased because the computation could be parallelized across many instances, and also avoided the central bottleneck of multiplying all moduli into one integer. Cost was also reduced, because cloud VM pricing was proportional to the *product* of compute time and RAM allocated. This means that monetary cost was already at least quadratic $O(n^2)$ with respect to the number of bits of input n, despite batch GCD being a quasilinear (time) algorithm. In addition, by splitting the computation into smaller parts, we reduced the cost's proportionality constant by enabling the use of lower-cost spare capacity such as preemptible instances.

4.4 Common Key Test

The common key test has essentially two steps. First, all of the certificates in the batch that have common keys must be detected. Second, each set of certificates that has common keys must be checked to see whether it is associated with a single device that obtained (or self-issued) a series of certificates with a single key. To accomplish the first stage, we implemented the `-common-key` option to the `cert_analyze` tool, which writes all of the certificates with a particular common key into a single output file.

To accomplish the second stage, we analyzed the certificate subject fields. Certificate data can be very noisy, and certificates that do not use proper 802.1AR formatting with (Product ID, Serial Number) are hard to interpret and unfortunately common. The formatting of the certificates we analyzed varied considerably even within the same vendor, and sometimes fields that should be present (such as the 802.1AR serial number) were absent.

We therefore developed vendor-specific code for the purpose of identifying devices through their manufacturing certificates, with coverage for many of the most common patterns. The vendor used 802.1AR (PID, SN) to identify its devices, and the most well-formed encoding was exemplified by a string such as

`PID:XY123-4BC SN:WX12-34YZ-A5B6`

in the subject serial number field. In other cases, the subject common name contained a PID and SN concatenated using a hyphen, for example:

`XY-1234-SEP000B4650ACA0`

where `XY-1234` represents the PID and the rest of the string after the second hyphen represents the SN. In yet other cases, the SN was not delimited from other data, but could be detected due to a predictable format:

`LLLYYWWSSSS`

where LLL are letters representing the manufacturing location, YYWW are a numeric encoding of the year and week, and SSSS are numbers or capital letters that represent a serial ID.

In addition to the PID and SN, we also made use of the ACT2 serial number, which can appear in an optional certificate extension. Cisco specifies an ASN.1 Object Identifier (OID) to represent the serial number of an ACT2 chip in the Subject Alternate Name field of a certificate[6]. The `cert_analyze` tool reports the OID and the associated value as in Figure 6. According to the specification, the value is a PrintableString with the format `ChipID=<chipSN>`, where `chipSN` is the BASE64-encoded ACT2 chip serial number.

There are several scenarios in which a single device may have been legitimately issued multiple certificates with the same key. One scenario is a "gold" device used during a pre-manufacturing testing process that repeatedly requests certificates from the production Certification Authority (CA), resulting in tens of thousands of certificates that have identical keys and subjects. Another scenario with identical keys and subjects is a reissuing of certificates before expiration (without changing the certificate public key). A

[6] ACT2 SUDI CA Certification Practice Statement, Cisco, 2014.

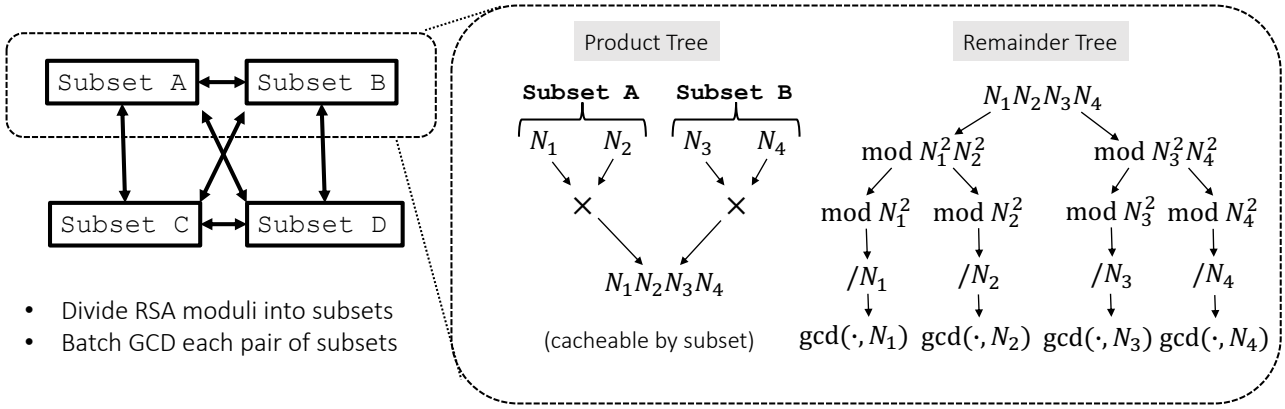

Figure 5: A cost-optimized batch GCD algorithm enables parallelization and scaling beyond RAM limits of single cloud instances, following Heninger et al. [11] and decoupling further than Hastings et al. [10].

```
{
   "other_name": {
   "type_id": "1.3.6.1.4.1.9.21.2.3",
   "value":
      "13334368697049443d5531524e53544977...41416e7875673d"
   }
}
```

Figure 6: Example of the ACT2 OID in a certificate (value truncated), represented as JSON by the `cert_analyze` tool.

trickier example is a product software upgrade that enables premium features. In this case, the device SN remains the same, but the PID can potentially change, usually by modifying one letter in the suffix of the PID to indicate that the device has been upgraded to enable premium functionality. This is a case where two certificates with different (PID, SN) pairs are legitimately associated with the same key. Fortunately, the vendor studied does not reuse SNs across product families, so the SN can still be used as an identifier for a single device, even when the PID changes slightly. Therefore, to search for common key entropy failures, we looked for certificates with the same public key but different serial numbers.

4.5 Exploitability

Common factors are easy to exploit, as the knowledge of one factor of an RSA public key makes it trivial to find the corresponding RSA private key[7]. Common keys cannot be exploited as directly.

To thoroughly exploit a product with a weak entropy source, an attacker should obtain an instance of the device, force it to generate many public key pairs, and record the private keys. By duplicating the environment used for manufacturing or device-issued certificates, the attacker can build up a set of typical private keys, which can then be used in a manner similar to a password dictionary attack. If a product suffers from LIE on successive reboots, then any cryptographic protocol it provides that uses key or nonce generation can be exploited. If it generates [EC]DSA signatures, a dictionary of r-values can be used to find its private signing key. If it generates a WiFi encryption key, or a VPN encryption key, then a dictionary of those keys can be used to passively decrypt traffic. We call these the **typical nonce** and **typical key** attacks, respectively. They would be especially damaging if an attacker could cause the victim's device to reboot, e.g. through resource exhaustion or some other vulnerability.

The effectiveness of a typical key attack is characterized as follows: an attacker that generates a set of typical keys of size k, then obtains a network session created immediately after bootup from a target device not under its control, will have probability p of being able to decrypt the session, because the target device's key will be in the typical set. The function $p(k)$ characterizes the vulnerability of the system, and is determined by the entropy of the source (under the reasonable assumption that the asymptotic equipartition property applies). Alternatively, a convenient characterization of the vulnerability may be to define $k_{\frac{1}{2}}$ as the number of keys such that $p(k_{\frac{1}{2}}) = \frac{1}{2}$. The choice of success probability is arbitrary but conventional.

It is an important and scientifically interesting open question to better understand $p(k)$ for different products. For some products, where a large number of duplicate keys were discovered, it is likely that a typical key attack would be very effective. For other products, only a few collisions occurred and the size of the typical set might run into the hundreds of millions or billions.

These attacks can be recast as a methodology for testing entropy, by developing a test harness that initiates restarts in a target device, then interacts with that device using a cryptographic protocol, and records the keys, nonces, and signatures. In the next section, we describe how we estimated the typical set.

4.6 Effective Entropy: Typical Set of Keys

Once weak entropy sources are discovered, it can be useful to quantify the effective entropy. In Section 4.5 above, we outlined a "typical keys" attack, in which an attacker generates the set of typical keys (by forcing many reboots). The important question about such attacks is: how big is the typical set? This can help prioritize the devices at greatest risk of compromise, as well as aid

[7]The `batch_gcd` tool can return the private key, for use in demonstrating the vulnerability.

in debugging and identifying the component where the entropy failure occurred.

We present a simple model, which can be applied to a single product in our results below (Table 4). Let K be the set of all keys, and $T \subset K$ be the set of typical keys. Let m denote the number of certificates, each of which contains a key that is generated at random (and thus will be in the typical set with high probability). Let d denote the number of distinct keys found, where $d \leq m$. Our goal is to estimate $|T|$.

One way to estimate the size of the typical set is to formulate it as the following combinatorial problem. Suppose there is an urn with an unknown number of balls (call it n), where each ball is known to be a different color. We draw with replacement k balls, and discover that the balls drawn cover only d distinct colors, where $d < k$. We would like to estimate n, the number of balls in the urn. Specifically, we would like the maximum likelihood estimate of n, i.e., the n that maximizes the probability of observing d distinct colors when making k draws with replacement.

4.6.1 Likelihood Function - Definition. In order to specify the likelihood function that we would like to maximize, we begin with some definitions. Define a "k-draw" to be an ordered sequence of k balls drawn with replacement from an urn. Further define a "d-color k-draw" to be a k-draw that is composed of exactly d distinct colors, where $d \leq k$. We can now define the likelihood function $L(n, k; d) = \Pr(d|n, k)$ to be the probability that a k-draw from an n-ball urn is also a d-color k-draw. It can be shown that:

$$L(n, k; d) = \Pr(d|n, k) = \frac{1}{n^k} \binom{n}{d} \sum_{i=0}^{d} (-1)^i \binom{d}{i} (d - i)^k.$$

In Appendix A and Appendix B, we give an intuitive example and proof of this expression. In Section 4.6.2, we hold k and d fixed, and determine the n that maximizes $L(n, k; d)$.

4.6.2 Maximum Likelihood Estimate. Recall that the objective was to estimate the size of the urn (n), given the observed data: k draws resulted in d distinct colors drawn ($d < k$). The maximum likelihood estimate (MLE) is defined as follows. Consider $L(n, k; d)$ as a function of n, with d and k fixed.

$$L(n, k; d) = \frac{1}{n^k} \binom{n}{d} \sum_{i=0}^{d} (-1)^i \binom{d}{i} (d - i)^k$$

We would like to determine the $n = \hat{n}$ that maximizes $L(n, k; d)$. For example, if $(d, k) = (95, 100)$, then $\hat{n} = 957$, as illustrated in Figure 7.

Finding the MLE involves a simplification step, followed by numerical optimization. Since the summation does not depend on n, maximizing $L(n, k; d)$ is equivalent to maximizing the function $f(n) = \frac{1}{n^k} \binom{n}{d}$. That is,

$$\hat{n} = \arg\max_n L(n, k, d) = \arg\max_n f(n) = \arg\max_n \frac{1}{n^k} \binom{n}{d}.$$

Although the binomial coefficient $\binom{n}{d}$ can be differentiated, enabling us to differentiate $f(n)$ and set it equal to zero, the resulting optimal n must satisfy

$$\frac{k}{n} = \sum_{i=0}^{d-1} \frac{1}{n - i}$$

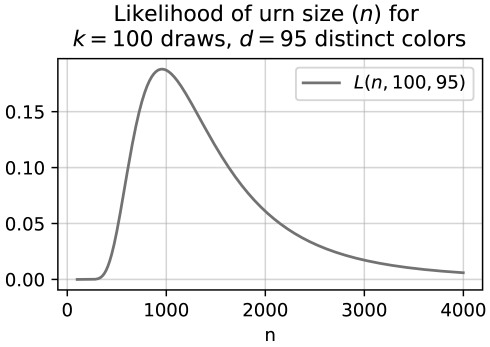

Figure 7: Likelihood vs urn size (n) for $k = 100$ draws and $d = 95$ distinct colors. The maximum likelihood estimate (MLE) for n is $\hat{n} = 957$.

Product ID	Product	Support Dates
CP-6901	Unified IP Phone 6901	Orderable
RV130W-A-K9-NA	RV130W VPN Router	EoS: 30-11-2024
WS-SVC-WISM2-K9	Wireless Services Module 2	EoS: 30-4-2022
AIR-CT5508-K9	5508 Wireless Controller	EoVSS: 31-7-2021
AIR-CT2504-K9	2504 Wireless Controller	EoVSS: 18-4-2021

Table 2: Products with weak entropy that were recently sold or supported, along with the relevant end-of-support dates.

which does not lend itself to a simple closed-form solution. Instead, $f(n)$ can be extended to the real numbers by writing it in terms of the gamma function $\Gamma(n)$, and then maximized by using Brent's algorithm to perform bounded optimization on $\ln f(n)$:

$$\ln f(n) = -k \ln n + \ln \Gamma(n + 1) - \ln \Gamma(n - d + 1) - \ln \Gamma(d + 1).$$

The optimum $\hat{n}$ value coincides well with intuition. When $d \ll k$, there are likely few balls in the urn, so $\hat{n} \approx d$. At the other extreme, when $d = k - 1$, the scenario is approximately a birthday problem where a single collision occurs, and $\hat{n} \approx k(k-1)/2$. In this latter case, coincidentally, the value of $L(\hat{n}, k, d)$ is close to $1/e$. Our formulation was easily implemented in a few lines of Python code that leverages the bounded optimization functionality provided by SciPy [22].

5 RESULTS

We found weak keys in certificates generated by several Cisco product families, including Wireless Local Area Network (WLAN), IP Telephony, Small Business Routers, and Linksys (both before and after its divestiture and sale to Belkin in 2013). Weak keys first started appearing in 2008, hit a peak of 250,000 per month in 2012, and have tapered off, with few appearing in 2020. Table 4 summarizes these findings, which include products not implicated in any prior studies. Most of the affected products were past their End-of-Support (EoS) or End-of-Vulnerability/Security Support (EoVSS) dates, but some were not, as summarized in Table 2 and detailed fully in Table 4. Note that these dates do not preclude a customer from purchasing an extended support contract.

From the manufacturing certificate dataset, we found common factors in ten distinct products, and common keys in many more. Most of the products that exhibited common factors also exhibited

Vendor	Certificates	RSA Keys	Factored Keys
Non-Cisco	37,625,299	29,110,765	2,700
Cisco	36,987	34,845	246
Total	37,662,286	29,145,610	2,946

Table 3: Summary of the Spring 2021 Sonar SSL certificates and RSA keys that appeared Feb 6–May 5, 2021. All entries represent counts of distinct certificates and keys.

common keys, which is strong evidence that the common keys are caused by weak entropy. For some products, the public key in a manufacturing certificate is generated by the device associated with it, while other products have keys that are generated by an external Hardware Security Module (HSM). All the products we found with weak keys had generated their own.

We also identified a **multiproduct key** that appears in the manufacturing certificates of nearly five million devices, across several different product families (CP-69xx, CP-89xx, CP-99xx, ATA-187, and Cius). These products are marked with a † in Table 4. The multiproduct key is an unusual case, because in most other common-key instances, only two devices share a particular key. Not every device in the multiproduct key set has the same key, and it appears that the more complex devices have more diversity of keys. This suggests that these products share a weak entropy source, and that its quality varies across products.

For the Spring 2021 internet scan dataset, Table 3 summarizes the findings of factorable RSA keys. The BGCD test discovered 2,632 factorable RSA keys in the Spring 2021 data; 246 were from Cisco products, and the majority of those were Cisco-Linksys. The remainder were from a wide variety of vendors, as detailed in Table 5. The three non-Linksys Cisco devices that exhibited common factors were are all Small Business Routers (see the last three rows of Table 4). Most of the Cisco PIDs did not exhibit weak keys. However, the data set contained fewer than 4000 certificates for each product ID, and the small size limits the effectiveness of the BGCD tests.

In addition, the common key test could not be applied to the Spring 2021 dataset. Similar to previous studies [11], TLS certificates retrieved from internet scans do not often contain device identity markers such as a serial number. Therefore, it was not possible to determine whether two certificates with the same key originated from the same device or different devices.

For all products whose manufacturing certificates exhibited weak keys in Table 4, we computed the maximum likelihood estimate for the size of the typical set of keys (see the column labeled $\log_2 |\text{TypicalSet}|$). For some products with the multiproduct key, half or more of the keys were common, and it is likely that a typical key attack would be very effective. For other products, such as the AIR-CT2504-K9, which had only 117 common keys in 451,417 certificates, the estimated size of the typical set was $2^{30.6} \approx 1.6$ billion.

6 DISCUSSION

Perhaps the most surprising result of this study was the discovery of the multiproduct key that was common across nearly five million devices (CVE-2022-20817). This discovery was made possible by

Product ID	Factorable Certs	Common-Key Certs	Total Certs	$\log_2 \lvert\text{TypicalSet}\rvert$	Dates
AIR-CT2504-K9	269	117	451417	30.6	2010-2020
AIR-CT5508-K9	617	93	256867	28.3	2008-2020
AIR-WLC4402-*-K9	8	9	73306	29.5	2006-2016
WS-SVC-WISM2-K9	30		19472	25.6	2010-2019
SVC-WiSM	6	18	40443	27.2	2006-2017
DMC250 (Linksys)	67	62	6370	19.7	2008-2009
DMP100 (Linksys)	87	139	7505	19.2	2008-2009
DMRW1000 (Linksys)	118	99	7343	19.4	2008-2009
PHM1200 (Linksys)	15		2043	20.0	2006-2007
VGA2000 (Linksys)	225	71	1033	13.4	2006-2007
C1100 (Aironet)		258	217623	27.5	2006-2015
C1130		2562	2464216	31.1	2006-2019
C1200		502	420772	28.4	2006-2015
C1240		2524	2521522	31.2	2006-2019
C1250		52	506938	32.2	2006-2019
C1310		216	191656	27.3	2006-2018
C1410		26	22553	24.2	2006-2016
C3201		23	24352	24.6	2006-2013
CP-7970		3	525683	36.0	2003-2016
DMC350		12	429	13.9	2008-2009
SVR200		2	2360	21.4	2007-2008
ATA-187†		245398	544356	20.0	2009-2014
CIUS-7/Cius/CiusSP†		39539	53137	15.7	2010-2012
CP-6901 †		219123	460401	19.6	2009-2020
CP-6911†		162965	193647	17.3	2009-2015
CP-6921†		1684990	3430898	22.5	2009-2014
CP-6922†		121	335	9.6	2011-2013
CP-6941†		782396	1545282	21.3	2009-2014
CP-6942†		87	324	10.1	2011-2013
CP-6945†		264462	813491	21.1	2010-2014
CP-6946†		193	549	10.4	2011-2013
CP-6951†		150	157	6.7	2010-2010
CP-6961†		145876	374045	19.7	2009-2014
CP-6962†		87	375	10.5	2011-2013
CP-8941†		87981	247542	19.2	2010-2014
CP-8945†		307134	1542140	22.8	2010-2015
CP-8961†		317261	951749	21.3	2009-2017
CP-9945†		258	261	7.4	2010-2010
CP-9951†		297769	480704	19.2	2009-2016
CP-9965†		394	410	8.1	2010-2010
CP-9971†		391091	695781	19.9	2008-2017
RV130W-A-K9-NA ‡	2	–	–	–	2021
RV120W ‡	7	–	–	–	2021
RV220W ‡	30	–	–	–	2021

Table 4: Cisco products with weak keys in devices certificates. "Factorable" means that two or more certificates had RSA public keys with common factors. "Common-key" means that multiple certificates had identical keys but distinct subjects, and appear to be distinct devices. A product ID marked with † exhibits certificates with the multiproduct key (Section 5). Certificates for product IDs marked with ‡ have been observed only in the device-issued data set (Section 4.1.2); dashes indicate missing data.

Figure 8: Cisco 6901 Unified IP Phone.

Figure 9: Cisco Small Business RV130n Wireless-N Multifunction VPN Router.

Figure 10: Cisco 2504 Wireless Controller.

Figure 11: Cisco 5508 Wireless Controller.

the common key test (Section 4.4). To our knowledge, this is the first study to test manufacturing certificates for unintentional key collisions across devices at such a large scale.

With the help of the product team, we investigated the most recently sold product affected by the multiproduct key: the CP-6901, a low-margin IP Phone that lacks a hardware entropy source. The device and its run-time operating system was FIPS-140 certified. However, for its first boot on the manufacturing floor, the device uses a different operating system—a diagnostic image (MontaVista 3.4.3 based on Linux kernel 2.6.10 for MIPS) responsible for self-tests and initialization, including key generation and the installation of a manufacturing certificate. Since the device does not have a hardware entropy source, the OS of the diagnostic image gathers entropy by observing the timing between network interrupts, and uses that entropy for RSA key generation. See Figure 13 for a pseudocode representation of this process. The design implies that when the device is first booted, it is absolutely critical that the network to which it is connected has sufficiently random traffic to seed the device random number generator. The intention was that a random traffic generator would be used on the network to supply entropy to devices during their initial boot.

Ultimately, the exact network environment that was used to initialize these devices could no longer be replicated at the time

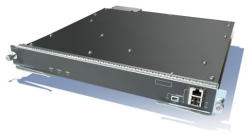

Figure 12: Cisco Wireless Services Module 2.

Weak Certs	Issuer: Organization Name
1524	TPLINK
700	(unknown)
227	DrayTek Corp.
211	Cisco-Linksys, LLC
194	HTTPS Mgmt Cert for SonicWALL
187	Tridium
106	Netgear Inc.
74	Kronos Incorporated
39	Cisco Systems, Inc.
37	D-Link
32	SAMSUNG
24	Technicolor
19	D-Link Corporation
15	Honeywell
11	Linksys International Inc.
11	Fortinet Ltd.
7	Advantech B+B SmartWorx s.r.o.
5	Archer C20
4	D-Link Taiwan
4	D-LINK
3	Huawei
3	Hewlett-Packard Company
2	HP
2	Gongjing
2	CalAmp Corp.
2	Primax
2	Alarm.com

Table 5: Vendors listed in weak certificates with factorable RSA keys.

of this study. However, the design of the system narrows the culprit to a number of possibilities: the network traffic generator may have failed or not been turned on, the switch connecting the traffic generator to the phone may have failed, or some other malfunction caused the network link to be either too quiet or too saturated to properly seed the device RNG. What *is* clear from this investigation, regardless of which was the ultimate cause, is that proper RSA key generation for this device depended on the composition of several systems that all had to function properly: the network environment, the adjacent network hardware, the device hardware, the operating system, and the key generation software. A failure in any of these systems could cause a catastrophic entropy failure and the generation of weak RSA keys.

Similarly, the CT2504 and CT5508 wireless controllers both used version 2.6.21 of the Linux Kernel, making it possible that they were affected by CVE-2007-2453. In this case, the devices had a hardware

```
// 1. Maintain a 256-byte entropy buffer.
INT8 *pEntropyData = &buffer[0]; // pointer: initially at buffer[0]

// 2. On packet arrival, if entropy buffer is not full, read a MIPS
//    system counter as a 4-byte cntVal using assembly:
asm volatile("mfc0 %0,$9; nop" : "=r"(count));

// 3. Add 4 bits of MIPS system counter into entropy pool
if (PKTCNT_IS_ODD) /* high 4 bits */
{
    *pEntropyData = ((INT8)(cntVal & 0x0000000f) << 4);
}
if (PKTCNT_IS_EVEN) /* low 4 bits */
{
    *pEntropyData = ((INT8)(cntVal & 0x0000000f);
    pEntropyData++; // advance to next byte of entropy buffer
}

// 4. Later, when key generation requires a number of random bytes,
//    entropy data generated earlier will be returned first.
```

Figure 13: Pseudocode illustrating the CP-6901 initial entropy generation based on network interrupts. During manufacturing, this occurs on first boot into a diagnostic image.

entropy source (Cavium), but only used that hardware in FIPS-140 mode. This means that a configuration oversight during the initial key generation phase may have caused the entropy failures for the CT2504 and CT5508. Fortunately, there exists a workaround: users may generate a Locally Significant Certificate (LSC) in FIPS mode, and use that instead of the Manufacturer Installed Certificate (MIC) for device identity and authentication.

Finally, the RV130W Small Business Routers were fully outsourced in their design and production. Since Cisco no longer had a relationship with the contract manufacturer at the time of this study, discovering the root cause of the entropy failure would have required prohibitive reverse engineering effort. In addition, the lack of good mechanisms to patch these products increased the number of applicable vulnerabilities. Moreover, certificates in this data set were publicly accessible through internet scans, putting the devices at an increased risk of compromise. For example, we discovered a pair of RV130W-A-K9-NA keys that shared a common factor. Given that there were only 519 RV130W-A-K9-NA certificates in the data set (and a total of 981 instances of RV130W-∗), this suggests a significant entropy weakness. The best mitigation for these routers is to avoid enabling the administrative portal on the WAN interface.

Two main recommendations arise from investigating the entropy failures revealed by our testing: (1) use hardware entropy sources, and (2) proactively test device public keys for proper entropy.

Using hardware entropy sources is becoming less costly: many modern CPUs and SoCs contain hardware entropy sources, including Intel and AMD through the RDRAND instruction, and the ARM-based Broadcom 2835 used in the Raspberry Pi. However, we recognize that for low-margin products, cost will always be an important factor. In addition, as demonstrated by the CT2504 and CT5508, the mere presence of a hardware entropy source may not be sufficient—the default configuration may disable it, e.g., for performance reasons. ACT2 and Aikido, the Cisco Trusted Platform Modules (TPMs), provide a hardware trust anchor, and can provide a seed for a software-based entropy pool, but they cannot directly provide entropy for products because they communicate across a slow serial bus. If performance is critical (and it often is), one solution is to seed software entropy from hardware, even if key generation is performed in software. This should be done whether the device is in FIPS-140 mode or not.

Nevertheless, the potential for mistakes underlines the need for our second recommendation: end-to-end testing of a large population of devices. The earlier in the product lifecycle that we can detect entropy issues, the better. Batch testing of device-generated keys could be proactively applied during pre-manufacturing testing, or during manufacturing. The tests we developed can be automated to enable continuous detection. Products with weak manufacturing certificates may have further weak entropy issues. For some products, further entropy testing may be warranted. One caveat is that batch testing the manufacturing certificates of products that use ACT2 provides no information about their entropy sources, because those key pairs are generated by a Safenet HSM. This is a very sound way to generate keys, but a new data collection effort is needed to test the entropy of these important products. Additional data, data sources and tests would broaden and strengthen the assurance provided. Data collection and analysis should continue, especially for device-issued certificates, and new tests such as SSH device keys, the typical key attack (Section 4.5) and repeated r-values in [EC]DSA signatures should be pursued.

7 LIMITATIONS

We discuss a number of limitations. First, an obvious restriction of the batch GCD algorithm for detecting common factors is that it applies only to RSA keys. Second, although the duplicate key detection methodology applies to any public key algorithm, a fundamental requirement is that it be possible to distinguish between distinct devices. That is, it must be possible to detect that two or more distinct devices are using the same key pair (which is concerning), as opposed to a single device that has been issued multiple certificates containing the same key (which is often acceptable). Identification of distinct devices is inherently vendor-specific; for the single vendor in our study, it was possible to write custom code to parse the 802.1AR device strings in order to identify product ID and serial number in X.509 certificates. This may not be possible for all vendors. Finally, root cause analysis for entropy failures can require substantial human effort, navigating organizational boundaries and different product teams. A device vendor's certification authority (CA) may be in a good position to observe keys across a large number of manufactured devices. We were fortunate that each identified entropy failure was followed by close collaboration between the CA, the vendor's PSIRT, and the product team to identify the cause and develop a mitigation. We recognize that organizational factors may contribute to the industry-wide challenge of detecting and preventing weak entropy in manufactured devices.

8 EFFICIENT POPULATION TESTING

Based on the authors' experience with the Cisco Certification Authority (CA), we offer some recommendations on how to conduct population-wide testing efficiently. Because millions of devices are not manufactured instantaneously, the Cisco CA provides a service that signs certificates as new devices are activated for the first

time during manufacturing. Thus, each night there exists a batch of certificates corresponding to the devices that were manufactured that day. By running our analysis periodically (nightly or weekly), it is possible to detect issues early in the production lifetime, so that the key generation process can be fixed for devices not yet manufactured, and software updates and mitigation procedures can be created for devices already sold to end-users.

As noted earlier, the common factor test is computationally intensive and limited to RSA. However, RSA is still in widespread use (e.g., in Cisco products), and needs to be addressed in practice. Fortunately, the computational cost is not prohibitive; our analysis of over 200 million certificates can be run every 3-4 days on GCP using one `m1-megamem-96` instance (totaling a few hundred dollars USD), a cost that is likely well within the reach of most device vendors. In fact, after the initial study was completed, the Cisco CA expressed preliminary interest in running the common key and common factor analysis on new certificates that are issued on a nightly basis. One way this can be done even more efficiently is to save an intermediate result of the previous batch GCD computations, which we sketch below. Suppose k certificates have already been checked, and j certificates have been created in the latest batch. If we stored the product $A = N_1 N_2 \cdots N_k$ of the already-checked moduli $(N_1, \ldots, N_k)$, then given the product A, checking the new moduli $(N_{k+1}, \ldots, N_{k+j})$ against themselves and against the existing set can be done as follows. Use the standard product tree to compute $B = N_{k+1} N_{k+2} \cdots N_{k+j}$; then compute the product $B' = A \cdot B$. Finally, use B' as the root of the remainder tree for checking $N_{k+1}, \ldots, N_{k+j}$. This can reduce the incremental work factor significantly. Asymptotically, this still amounts to an $O(n^2)$ computation over time, but the constant is much smaller, and in practice there are useful ways to limit the size of n: check subsets of certificates from related product families, and age-out devices that are end-of-life.

9 ETHICAL CONSIDERATIONS

We worked closely with the vendor PSIRT to investigate vulnerabilities found in products and coordinated the release of this work with the timing of advisory publications to affected customers. For the public internet scans, we also contacted the third party vendors that appeared in the issuer fields of certificates that exhibited weak keys. Some vendors responded positively: for example, after we communicated our methodology, DrayTek quickly replicated the weak key findings with an internal population test and developed firmware mitigations. Advantech traced their weak keys to End-of-Life devices without hardware RNGs, and published a security advisory with suggested workarounds [2].

10 CONCLUSION

A good entropy source is critical for cryptography, and this study demonstrates that a combination of the Low Initial Entropy (LIE) problem and misconfiguration of manufacturing processes has resulted in the generation of weak keys in millions of device certificates in the last decade. Detection of weak entropy best accomplished through population testing of purportedly random values in the final output or shipping state of a product. Testing a single device repeatedly will not reveal LIE issues. Neither will unit tests suffice: a bug in any stage of device manufacturing can result in entropy failure, whether the bug is in hardware, software, configuration, or a composition of manufacturing stages. Fortunately, a large-scale end-to-end testing strategy can help detect entropy failures, even when devices originate from heterogenous manufacturing environments, and even when not all devices have hardware random number generators (especially lower-margin devices). Moreover, once an entropy weakness is discovered, entropy pool size estimation can help a vendor debug the root cause, as well as estimate the effort required for an attacker to exploit the weakness. A vendor's certification authority (CA) is often well-positioned to observe keys across a large product portfolio. Thus, we recommend a collaboration between a vendor CA, the product teams, and a vendor's PSIRT in order to provide early detection and perhaps prevention of large scale entropy failures in future network devices.

ACKNOWLEDGMENTS

We gratefully thank Eric Hampshire and Anita Shah for providing us with a complete dump of the many certificates that the Cisco certification authority has issued, Bill Sulzen for help understanding the processes around certificate issuance, and Blake Anderson for providing certificates from Cisco's operational network. Ann Chen, Michael Schueler, and Dario Ciccarone of Cisco PSIRT coordinated the investigation of these vulnerabilities. And numerous members of product teams helped provide deep dives into their products' internals in order to investigate their entropy generation mechanisms: Shijie Zhang, Carrie Wu, Steve Yu, Karrthik Venu, Channamallikarjuna Patil, Satyanarayana Yara.

REFERENCES

[1] IEEE 802.1AR. 2018. IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity. *IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009)* (2018). https://doi.org/10.1109/IEEESTD.2018.8423794

[2] Advantech. 2023. *Security Advisory SA-2023-01: v2 Products May Generate Insufficiently Random Keys.* Retrieved October 24, 2023 from https://icr.advantech.cz/support/router-models/download/511/sa-2023-01-insufficient-randomness.pdf

[3] Daniel J. Bernstein. 2004. How to find smooth parts of integers. *http://cr.yp.to/papers.html#smoothparts* (2004).

[4] Daniel J. Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko van Someren. 2013. Factoring RSA Keys from Certified Smart Cards: Coppersmith in the Wild. In *Advances in Cryptology - ASIACRYPT 2013*. Springer Berlin Heidelberg, Berlin, Heidelberg, 341–360.

[5] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. 2014. Elliptic Curve Cryptography in Practice. In *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, Berlin, Heidelberg, 157–175.

[6] Enrico Branca, Farzaneh Abazari, Ronald Rivera Carranza, and Natalia Stakhanova. 2021. Origin Attribution of RSA Public Keys. In *Security and Privacy in Communication Networks – 17th EAI International Conference (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 398)*. Springer. https://doi.org/10.1007/978-3-030-90019-9_19

[7] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. 2018. Where Did I Leave My Keys? Lessons from the Juniper Dual EC Incident. *Commun. ACM* 61, 11 (Oct 2018), 148–155. https://doi.org/10.1145/3266291

[8] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and Tim Polk. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280. IETF. http://tools.ietf.org/rfc/rfc5280.txt

[9] Brian Dawkins. 1991. Siobhan's Problem: The Coupon Collector Revisited. *The American Statistician* 45, 1 (1991).

[10] Marcella Hastings, Joshua Fried, and Nadia Heninger. 2016. Weak Keys Remain Widespread in Network Devices. In *ACM Internet Measurement Conference*. http://dl.acm.org/citation.cfm?id=2987486

[11] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2012. Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. In *21th USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger
[12] Jonathan Kilgallin and Ross Vasko. 2019. Factoring RSA Keys in the IoT Era. In *IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications*.
[13] Eric Langford and Rebecca Langford. 2002. Solution of the Inverse Coupon Collector's Problem. *The Mathematical Scientist* 27 (2002).
[14] David McGrew, Blake Anderson, Scott Fluhrer, and Chris Shenefiel. 2017. PRNG Failures and TLS Vulnerabilities in the Wild. In *Real World Crypto (RWC)*.
[15] James D. Nichols, Barry R. Noon, S. Lynne Stokes, and James E. Hines. 1981. Remarks on the Use of Mark-Recapture Methodology in Estimating Avian Population Size. *Studies in Avian Biology* 6 (1981).
[16] National Institute of Standards and Technology. 2001. Security Requirements for Cryptographic Modules. Federal Information Processing Standards Publications (FIPS PUBS) 140-2. Change Notice 2 December 03, 2002 (2001). https://doi.org/10.6028/nist.fips.140-2
[17] National Institute of Standards and Technology. 2003. Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program. https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402IG.pdf. Last Update March 17, 2023 (2003).
[18] National Institute of Standards and Technology. 2019. Security Requirements for Cryptographic Modules. Federal Information Processing Standards Publications (FIPS PUBS) 140-3. Published March 22, 2019 (2019). https://doi.org/10.6028/NIST.FIPS.140-3
[19] Richard P. Stanley. 2011. *Enumerative Combinatorics* (2nd ed.). Cambridge Studies in Advanced Mathematics, Vol. 1. Cambridge University Press. https://doi.org/10.1017/CBO9781139058520
[20] Petr Svenda, Matús Nemec, Peter Sekan, Rudolf Kvasnovský, David Formánek, David Komárek, and Vashek Matyás. 2016. The Million-Key Question – Investigating the Origins of RSA Public Keys. In *25th USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/svenda
[21] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, and Mike Boyle. 2018. Recommendation for the Entropy Sources Used for Random Bit Generation. National Institute of Standards and Technology Special Publication (SP) 800-90B. (2018). https://doi.org/10.6028/NIST.SP.800-90B
[22] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

A LIKELIHOOD FUNCTION - SMALL EXAMPLE

In Section 4.6.1, we claimed that the following expression represents the probability that a k-draw from an n-ball urn is also a d-color k-draw:

$$L(n,k;d) = \Pr(d|n,k) = \frac{1}{n^k}\binom{n}{d}\sum_{i=0}^{d}(-1)^i\binom{d}{i}(d-i)^k.$$

Before proving the general case in Appendix B, we build intuition with a small example: $n = 10$, $k = 5$, $d = 3$. The number of 5-draws (of any colors) from a 10-ball urn is 10^5. To compute $\Pr(d = 3|n = 10, k = 5)$, we also need to know how many of the 5-draws are compromised of exactly 3 distinct colors, i.e., how many are 3-color 5-draws.

Enumerating the 3-color 5-draws from a 10-ball urn can be conceptually broken up into two steps.

(1) Choose 3 out of the 10 colors.
(2) Generate all 3-color 5-draws using only the 3 chosen colors.

The first step has $\binom{10}{3}$ possibilities, and the second step can be computed using the principle of inclusion-exclusion in the following manner. Let S be the set of all 5-draws that use only the 3 chosen colors from the first step (but not necessarily all 3 colors). Since each draw can be any of the three colors, $|S| = 3^5$. For $i \in \{1,2,3\}$, let Q_i be the subset of S that does *not* contain color i. The cardinalities of the Q_i and their intersections are:

- $|Q_i| = (3-1)^5 = 32$ for all i. These are the 5-draws in S that do *not* use the color i.
- $\left|Q_i \cap Q_j\right| = (3-2)^5 = 1$ for all $i \neq j$. These are the 5-draws in S that use *neither* color i nor color j.
- $|Q_1 \cap Q_2 \cap Q_3| = (3-3)^5 = 0$. There are no 5-draws in S that use none of the 3 chosen colors, since S is restricted to those colors.

We wish to compute $\left|\overline{Q_1} \cap \overline{Q_2} \cap \overline{Q_3}\right|$, i.e., the number of 5-draws in S that actually use all 3 colors. This can done by applying the principle of inclusion-exclusion. (Section B gives a concise statement in terms of Stirling numbers of the second kind, but inclusion-exclusion provides the combinatorial basis of the formula.)

$$\begin{aligned}
\left|\overline{Q_1} \cap \overline{Q_2} \cap \overline{Q_3}\right| &= |S| - \left|\overline{\overline{Q_1} \cap \overline{Q_2} \cap \overline{Q_3}}\right| \\
&= |S| - |Q_1 \cup Q_2 \cup Q_3| \\
&= |S| - \sum_i |Q_i| + \sum_{i\neq j}\left|Q_i \cap Q_j\right| - |Q_1 \cap Q_2 \cap Q_3| \\
&= |S| - \binom{3}{1}|Q_1| + \binom{3}{2}|Q_1 \cap Q_2| - |Q_1 \cap Q_2 \cap Q_3| \\
&= \binom{3}{0}3^5 - \binom{3}{1}2^5 + \binom{3}{2}1^5 - \binom{3}{3}0^5 \\
&= 150.
\end{aligned}$$

The number of 3-color 5-draws from a 10-ball urn is $\binom{10}{3} \cdot 150 = 18000$. Therefore, we can compute the likelihood:

$$\begin{aligned}
L(n = 10, k = 5; d = 3) &= \frac{|\text{3-color 5-draws from 10-ball urn}|}{|\text{5-draws from 10-ball urn}|} \\
&= \frac{\binom{10}{3} \cdot 150}{10^5} = \frac{18000}{10^5} = \frac{9}{50}.
\end{aligned}$$

The computation above assumes that the size of the urn is known ($n = 10$). Section 4.6.2 will remove that assumption, and instead maximize $L(n,k;d)$ as a function of n.

B LIKELIHOOD FUNCTION - GENERAL CASE

We now derive the general expression for the likelihood function $L(n,k;d)$. By definition,

$$L(n,k;d) = \Pr(d|n,k) = \frac{|\{d\text{-color } k\text{-draws from } n\text{-ball urn}\}|}{|\{k\text{-draws from } n\text{-ball urn}\}|}$$

The denominator is simply n^k. The numerator can be computed in two steps:

(1) Choose d out of the n colors: $\binom{n}{d}$ possibilities.
(2) Generate all d-color k-draws using only the d chosen colors.

The second step is related to a well-known combinatorial problem [19].[8] Specifically, there is a close relationship between:

[8] See Richard Stanley's Enumerative Combinatorics, Vol. 1, 2nd ed., Section 1.9 The Twelvefold Way. One problem in the catalog of twelve involves counting surjective functions $f : N \to X$ for finite sets N and X. Stirling numbers of the second kind arise from this problem.

- The Stirling number of the second kind $S(k, d)$, i.e., the number of ways to partition a set of k elements into d non-empty *unlabeled* subsets.
- The number of d-color k-draws that use a fixed set of d colors.

We relate these two problems as follows. Consider each of the k draws (indexed $1, 2, \ldots, k$) as an element, and each of the d colors as the label for a subset. If the m-th draw is a ball of color i, then we assign element m to subset i. A d-color k-draw corresponds to a partition of k elements into d *labeled* subsets. The only difference between this and the $S(k, d)$ scenario is whether the subsets are labeled or unlabeled. Since all permutations of labels produce distinct partitions, the difference is a factor of $d!$, so the number of d-color k-draws using only d chosen colors is $d! \cdot S(k, d)$. The expression for $d! \cdot S(k, d)$ is:

$$d! \cdot S(k, d) = \sum_{i=0}^{d} (-1)^i \binom{d}{i} (d - i)^k.$$

In Appendix A, we arrived at an instance of this expression via inclusion-exclusion. There also exist non-combinatorial proofs [19]. Combining all of the steps above, we can derive a general expression for the likelihood function.

$$\begin{aligned} L(n, k; d) = \Pr(d|n, k) &= \frac{|\{d\text{-color } k\text{-draws from } n\text{-ball urn}\}|}{|\{k\text{-draws from } n\text{-ball urn}\}|} \\ &= \frac{1}{n^k} \cdot |\{d\text{-color } k\text{-draws from } n\text{-ball urn}\}| \\ &= \frac{1}{n^k} \cdot \binom{n}{d} \cdot (d! \cdot S(k, d)) \\ &= \frac{1}{n^k} \binom{n}{d} \sum_{i=0}^{d} (-1)^i \binom{d}{i} (d - i)^k. \end{aligned}$$