Biased RSA Private Keys: Origin Attribution of GCD-Factorable Keys

Adam Janovsky[1,2(✉)], Matus Nemec[3], Petr Svenda[1], Peter Sekan[1], and Vashek Matyas[1]

[1] Masaryk University, Brno, Czech Republic
adamjanovsky@mail.muni.cz
[2] Invasys, Brno, Czech Republic
[3] Linköping University, Linköping, Sweden

Abstract. In 2016, Švenda et al. (USENIX 2016, The Million-key Question) reported that the implementation choices in cryptographic libraries allow for qualified guessing about the origin of public RSA keys. We extend the technique to two new scenarios when not only public but also private keys are available for the origin attribution – analysis of a source of GCD-factorable keys in IPv4-wide TLS scans and forensic investigation of an unknown source. We learn several representatives of the bias from the private keys to train a model on more than 150 million keys collected from 70 cryptographic libraries, hardware security modules and cryptographic smartcards. Our model not only doubles the number of distinguishable groups of libraries (compared to public keys from Švenda et al.) but also improves more than twice in accuracy w.r.t. random guessing when a single key is classified. For a forensic scenario where at least 10 keys from the same source are available, the correct origin library is correctly identified with average accuracy of 89% compared to 4% accuracy of a random guess. The technique was also used to identify libraries producing GCD-factorable TLS keys, showing that only three groups are the probable suspects.

Keywords: Cryptographic library · RSA factorization · Measurement · RSA key classification · Statistical model

1 Introduction

The ability to attribute a cryptographic key to the library it was generated with is a valuable asset providing direct insight into cryptographic practices. The slight bias found specifically in the primes of RSA private keys generated by the OpenSSL library [14] allowed to track down the devices responsible for keys found in TLS IPv4-wide scans that were in fact factorable by distributed GCD algorithm. Further work [23] made the method generic and showed that many

Full details, datasets and paper supplementary material can be found at https://crocs.fi.muni.cz/papers/privrsa_esorics20.

L. Chen et al. (Eds.): ESORICS 2020, LNCS 12309, pp. 505–524, 2020.
https://doi.org/10.1007/978-3-030-59013-0_25

other libraries produce biased keys allowing for the origin attribution. As a result, both separate keys, as well as large datasets, could be analyzed for their origin libraries. The first-ever explicit measurement of cryptographic library popularity was introduced in [18], showing the increasing dominance of the OpenSSL library on the market. Furthermore, very uncommon characteristics of the library used by Infineon smartcards allowed for their entirely accurate classification. Importantly, this led to a discovery that the library is, in fact, producing practically factorable keys [19]. Consequently, more than 20 million of eID certificates with vulnerable keys were revoked just in Europe alone. The same method allowed to identify keys originating from unexpected sources in Estonian eIDs. Eventually, the unexpected keys were shown to be injected from outside instead of being generated on-chip as mandated by the institutional policy [20].

While properties of RSA primes were analyzed to understand the bias detected in public keys, no previous work addressed the origin attribution problem *with* the knowledge of private keys. The reason may sound understandable – while the public keys are readily available in most usage domains, the private keys shall be kept secret, therefore unavailable for such scrutiny. Yet there are at least two important scenarios for their analysis: 1) Tracking sources of GCD-factorable keys from large TLS scans and 2) a forensic identification of black-box devices with the capability to export private keys (e.g., unknown smartcard, remote key generation service, or in-house investigation of cryptographic services). The mentioned case of unexpected keys in Estonian eIDs [20] is a practical example of a forensic scenario, but with the use of public keys only. The analysis based on private keys can spot even a smaller deviance from the expected origin as the bias is observed closer to the place of its inception. This work aims to fill this gap in knowledge by a careful examination of both scenarios.

We first provide a solid coverage of RSA key sources used in the wild by expanding upon the dataset first released in [23]. During our work, we more than doubled the number of keys in the dataset, gathered from over 70 distinct cryptographic software libraries, smartcards, and hardware security modules (HSMs). Benefiting from 158.8 million keys, we study the bias affecting the primes p and q. We transform known biased features of public keys to their private key analogues and evaluate how they cluster sources of RSA keys into groups. We use the features in multiple variants of Bayes classifier that are trained on 157 million keys. Subsequently, we evaluate the performance of our classifiers on further 1.8 million keys isolated from the whole dataset. By doing so, we establish the reliability results for the forensic case of use, when keys from a black-box system are under scrutiny. On average, when looking at just a single key, our best model is able to correctly classify 47% of cases when all libraries are considered and 64.6% keys when the specific sub-domain of smartcards is considered. These results allow for much more precise classification compared to the scenario when only public keys are available.

Finally, we use the best-performing classification method to analyze the dataset of GCD-factorable RSA keys from the IPv4-wide TLS scan collected by Rapid7 [21].

The main contributions of this paper are:

- A systematic mapping of biased features of RSA keys evaluated on a more exhaustive set of cryptographic libraries, described in Sect. 2. The dataset (made publicly available for other researchers) lead to 26 total groups of libraries distinguishable based on the features extracted from the value of RSA private key(s).
- Detailed evaluation of the dataset on Bayes classifiers in Sect. 3 with an average accuracy above 47% where only a single key is available, and almost 90% when ten keys are available.
- An analysis of the narrow domain of cryptographic smartcards and libraries used for TLS results in an even higher accuracy, as shown in Sect. 4.
- Practical analysis of real-world sources of GCD-factorable RSA keys from public TLS servers obtained from internet-wide scans in Sect. 5.

The paper roadmap has been partly outlined above, Sect. 7 then shows related work and Sect. 8 concludes our paper.

2 Bias in RSA Keys

Various design and implementation decisions in the algorithms for generating RSA keys influence the distributions of produced RSA keys. A specific type of bias was used to identify OpenSSL as the origin of a group of private keys [17]. Systematic studies of a wide range of libraries [18,23] described more reasons for biases in RSA keys in a surprising number of libraries. In the majority of cases, the bias was not strong enough to help factor the keys more efficiently. Previous research [23] identified multiple sources of bias that our observations from a large dataset of private RSA keys confirm:

1. **Performance optimizations**, e.g., most significant bits of primes set to a fixed value to obtain RSA moduli of a defined length.
2. **Type of primes**: probable, strong, and provable primes:
   - For probable primes, whether candidate values for primes are chosen randomly or a single starting value is incremented until a prime is found.
   - When generating candidates for probable primes, small factors are avoided in the value of $p-1$ by multiple implementations without explaining.
   - Blum integers are sometimes used for RSA moduli – both RSA primes are congruent to 3 modulo 4.
   - For strong primes, the size of the auxiliary prime factors of $p-1$ and $p+1$ is biased.
   - For provable primes, the recursive algorithm can create new primes of double to triple the binary length of a given prime; usually one version of the algorithm is chosen.
3. **Ordering of primes**: are the RSA primes in private key ordered by size?
4. **Proprietary algorithms**, e.g., the well-documented case of Infineon fast prime key generation algorithm [19].

5. **Bias in the output of a PRNG**: often observable only from a large number of keys from the same source;
6. **Natural properties of primes** that do not depend on the implementation.

2.1 Dataset of RSA Keys

We collected, analyzed, and published the largest dataset of RSA keys with a known origin from 70 libraries (43 open-source libraries, 5 black-box libraries, 3 HSMs, 19 smartcards). We both expanded the datasets from previous work [18,23] and generated new keys from additional libraries for the sake of this study. We processed the keys to a unified format and made them publicly available. Where possible, we analyzed the source code of the cryptographic library to identify the basic properties of key generation according to the list above.

We are primarily interested in 2048-bit keys, what is the most commonly used key length for RSA. As in previous studies [18,23], we also generate shorter keys (512 and 1024 bits) to speed up the process, while verifying that the chosen biased features are not influenced by the key size. This makes the keys of different sizes interchangeable for the sake of our study. We assume that repeatedly running the key generation locally approximates the distributed behaviour of many instances of the same library. This model is supported by the measurements taken in [18] where distributions of keys collected from the Internet exhibited the same biases as locally generated keys.

2.2 Choice of Relevant Biased Features

We extended the features used in previous work on public keys to their equivalent properties of private keys:

Feature '5p and 5q': Instead of the most significant bits of the modulus, we use five most significant bits of the primes p and q. The modulus is defined by the primes, and the primes naturally provide more information. We chose 5 bits based on a frequency analysis of high bits. Further bits are typically not biased and reducing the size of this feature prevents an exponential growth of the feature space.

Feature 'blum': We replaced the feature of second least significant bit of the modulus by the detection of Blum integers. Blum integers can be directly identified using the two prime factors. When only the modulus is available, we can rule out the usage of Blum integers, but not confirm it.

Feature 'mod': Previous work used the result of modulus modulo 3. It was known that primes can be biased modulo small primes (due to avoiding small factors of $p-1$ and $q-1$). The authors only used the value 3, because it is possible to rule out that 3 is being avoided as a factor of $p-1$, when the modulus equals 2 modulo 3 [23]. It is not possible to rule out higher factors from just a single modulus. With the access to the primes we can directly check for this bias for all factors. We detected four categories of such bias, each avoiding all small odd

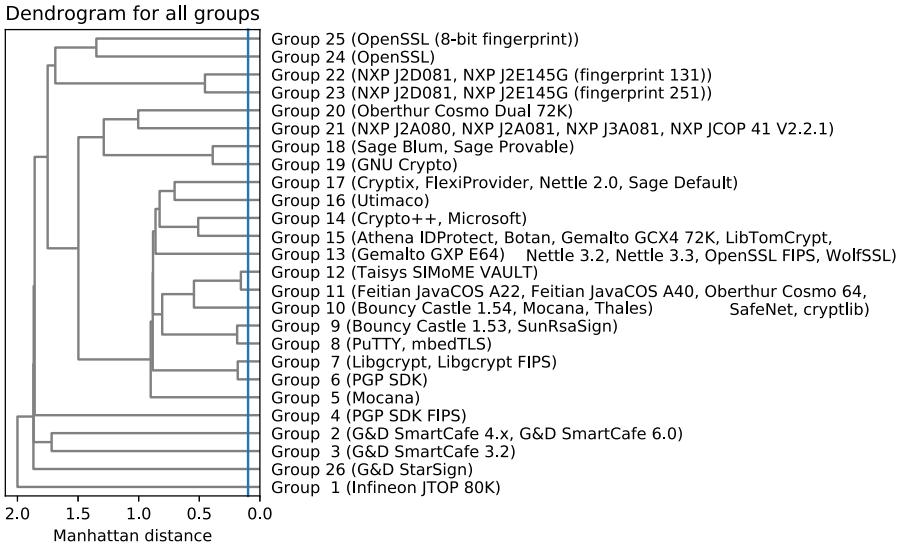

Fig. 1. How the keys from various libraries differ can be depicted by a dendrogram. It tells us, w.r.t. our feature set, how far from each other the probability distributions of the sources are. We can then hierarchically cluster the sources into groups that produce similar keys. The blue line at 0.085 highlights the threshold of differentiating between two sources/groups. This threshold yields 26 groups using our feature set.

prime factors up to a threshold. We use these categories directly by looking at small odd divisors of $p-1$ and $q-1$ and note if none were detected: 1) up to 17863, 2) up to 251, 3) up to 5, 4) none – at least one value is divisible by 3.

Feature 'roca': We use a specific fingerprint of factorable Infineon keys published in [19].

2.3 Clustering of Sources into Groups

Since it is impossible to distinguish sources that produce identically distributed keys, we introduce a process of clustering to merge similar sources into groups. We cluster two sources together if they appear to be using identical algorithms based on the observation of the key distributions. We measure the difference in the distributions using the Manhattan distance[1]. The absolute values of the distances depend on the actual distributions of the features. Large distances correlate with significant differences in the implementations. Note, that very

[1] We experimented with Euclidean distance and fractional norms. While Euclidean distance is a proper metric, our experiments showed that it is more sensitive to the noise in the data, creating separable groups out of sources that share the same key generation algorithms. On the other hand, fractional norms did not highlight differences between sources that provably differ in the key generation process.

small observed distances may be only the result of noise in the distributions instead of a real difference, e.g., due to a smaller number of keys available.

We attempt to place the clustering threshold as low as possible, maximizing the number of meaningful groups. If we are not able to explain why two clusters are separated based on the study of the algorithms and distributions of the features, the threshold needs to be moved higher to join these clusters. We worked with distributions that assume all features correlated (as in [23]).

The resulting classification groups and the dendrogram is shown in Fig. 1. We placed the threshold value at 0.085. By moving it higher than to 0.154, we would lose the ability to distinguish groups 11 and 12. It would be possible to further split group 14, as there is a slight difference in the prime selection intervals used by Crypto++ and Microsoft [23]. However, the difference manifests less than the level of noise in other sources, requiring the threshold to be put at 0.052, what would create several false groups. We use the same clustering throughout the paper, although the value of the threshold would change when the features change. Note that different versions of the same library may fall into different groups, mostly because of the algorithm changes between these versions. This, for instance, is the case of the Bouncy Castle 1.53, and 1.54.

3 Model Selection and Evaluation

How accurately we can classify the keys depends on several factors, most notably on: the libraries included in the training set, number of keys available for classification, features extracted from the classified keys, and on the classification model. In this section, we focus on the last factor.

3.1 Model Selection

As generating the RSA keys is internally a stochastic process, we choose the family of probabilistic models to address the source attribution problem. Since there is no strong motivation for complex machine learning models, we utilize simple classifiers. More sophisticated classifiers could be built based on our findings when the goal is to reach higher accuracy or to more finely discriminate sources within a group. The rest of this subsection describes the chosen models.

Naïve Bayes Classifier. The first investigated model is a *naïve Bayes classifier*, called naïve because it assumes that the underlying features are conditionally independent. Using this model, we apply the maximum-likelihood decision rule and predict the label as $\hat{y} = \operatorname{argmax}_y P(X = x \mid y)$. Thanks to the naïve assumption, we may decompose this computation into $\hat{y} = \operatorname{argmax}_y \prod_{i=1}^{n} P(x_i \mid y)$ for the feature vector $x = (x_1, \ldots, x_n)$.

Bayes Classifier. We continue to develop the approach originally used in [23] that used the *Bayes classifier* without the naïve assumption. Several reasons motivate this. First, it allows to evaluate how much the naïve Bayes model suffers from the violated independence assumption (on this specific dataset). Secondly,

it enables us to access more precise probability estimates that are needed to classify real-world GCD-factorable keys. Additionally, we can directly compare the classification accuracy of private keys with the case of the public keys from [23]. However, one of the main drawbacks of the Bayes classifier is that it requires exponentially more data with the growing number of features. Therefore, when striving for high accuracy achievable by further feature engineering, one should consider the naïve Bayes instead.

Naïve Bayes Classifier with Cross-Features. The third investigated option is the naïve Bayes classifier, but we merged selected features that are known to be correlated into a single feature. In particular, we merged the features of the most significant bits (of p, q) into a single cross-feature. Subsequently, the naïve Bayes approach is used. This enables us to evaluate whether merging clearly interdependent features into one will affect the performance of naïve Bayes classifier w.r.t. this specific dataset.

3.2 Model Evaluation

Methodology of Classification and Metrics. Our training dataset contains 157 million keys and the test set contains 1.8 million keys. We derived the test set by discarding 10 thousand keys of each source from the complete dataset before clustering. This assures that each group has the test set with at least 10 thousand keys. Accordingly, since the groups differ in the number of sources involved, the resulting test dataset is imbalanced. For this reason, we employ the metrics of precision and recall when possible. However, we represent the model performance by accuracy measure in the tables and in more complex classification scenarios.

For *group X*, the precision can be understood as a fraction of correctly classified keys from *group X* divided by the number of keys that were marked as *group X* by our classifier. Similarly, the recall is a fraction of correctly classified keys from *group X* divided by a total number of keys from *group X* [11]. We also evaluate the performance of the models under the assumption that the user has a *batch* of several keys from the same source at hand. This scenario can arise, e.g., when a security audit is run in an organization and all keys are being tested. Furthermore, to react to some often misclassified groups, we additionally provide the answer "this key originates from *group X* or *group Y*" to the user (and we evaluate the confidence of these answers).

Table 1. Performance comparison of different models on the dataset with all libraries. Note that the precision of a random guess classifier is 3.8% when considering 26 groups.

Model	Avg. precision	Avg. recall
Bayes classifier	43.2%	47.6%
Naïve Bayes classifier	40.9%	46.2%
Cross-feature naïve B	41.7%	47.6%

Comparison of the Models. The overall comparison of all three models can be seen in Table 1. If the precision for some group is undefined, i.e., no key is allegedly originating from this group, we say that the precision is 0. We evaluate the naïve Bayes classifier on the same features that were used for Bayes classifier to measure how much classification performance is lost by introducing the feature independence assumption. A typical example of interdependent features is that the most significant bits of primes p and q are intentionally correlated to preserve the expected length of the resulting modulus n. Pleasantly, the observed precision (recall) decrease is only 2.3% (1.4%) when compared to the Bayes classifier. Accordingly, this suggests that a larger number of different features than usable with the Bayes classifier (due to exponential growth in complexity) can be considered when the naïve Bayes classifier is used. As a result, further improvement of the performance might be achieved, despite ignoring the dependencies among features. Overall, the Bayes classifier shows the best results. When a single key is classified, the average success rate for the 26 groups is captured by precision of 43.2% and a recall of 47.6%. Still, there is a wide variance between the performance in specific groups. A detailed table of results together with a discussion is presented in Appendix A.

4 Classification with Prior Information

Section 2 outlined the process of choosing a threshold value that determines the critical distance for distinguishing between distinct groups. Inevitably, the same threshold value directly influences the number of groups after the clustering task. As such, the threshold introduces a trade-off between the model performance and the number of discriminated groups. The smaller the difference between group distributions is, the more they are similar, and the model performance is lower as more misclassification errors occur. The objective of this section is to examine the classification scenario when some prior knowledge is available to the analyst, limiting the origin of keys to only a subset of all libraries or increase the likelihood of some. Since Sect. 3 showed that the Bayes classifier provides the best performance, this chapter considers only this model.

Prior knowledge can be introduced into the classification process in multiple ways, e.g., by using a prior probability vector that considers some groups more prevalent. We also note that the measurement method of [18] can be used to obtain such prior information, but a relatively large dataset (around 10^5 private keys) is required that may not be available. Our work, therefore, considers a different setting when some sources of the keys are ruled-out *before* the classifier is constructed. Such scenario arises e.g., when the analyst knows that the scrutinized keys were generated in an unknown cryptographic smartcard. In such case, HSMs and other sources of keys can thus be omitted from the model altogether what will arguably increase the performance of the classification process. Another example is leaving out libraries that were released after the classified data sample was collected.

Table 2. Bayes classifier performance on three analyzed partitionings of the dataset – complete dataset with all libraries (*All libraries*), smartcards only (*Smartcards domain*), libraries and HSMs expected to be used for TLS (*TLS domain*) and specific subset of TLS domain where only single prime is available due to the nature of results obtained by GCD factorization method (*Single-prime TLS domain*). Comparison with the random guess as a baseline is provided (here, accuracy equals precision and recall).

Dataset	Avg. precision	Avg. recall	Random guess (baseline)
All libraries	43.2%	47.6%	3.8%
Smartcards domain	61.9%	64.6%	8.3%
TLS domain	45.5%	42.2%	7.7%
Single-prime TLS domain	28.8%	36.2%	11.1%

We present the classification performance results for three scenarios with a limited number of sources – 1) cryptographic smartcards (Sect. 4.1), 2) sources likely to be used in the TLS domain (Sect. 4.2) and 3) a specific case of GCD-factorable keys from the TLS domain, where only one out of two primes can be used for classification (see Sect. 4.3 for more details). The comparison of models for these scenarios can be seen in Table 2.

To compute these models we first, discard the sources that cannot be the origin of the examined keys according to the prior knowledge of the domain (e.g., smartcards are not expected in TLS). Next, we re-compute the clustering task to obtain fewer groups than on the dataset with all libraries. Finally, we compute the classification tables for the reduced domain and evaluate the performance.

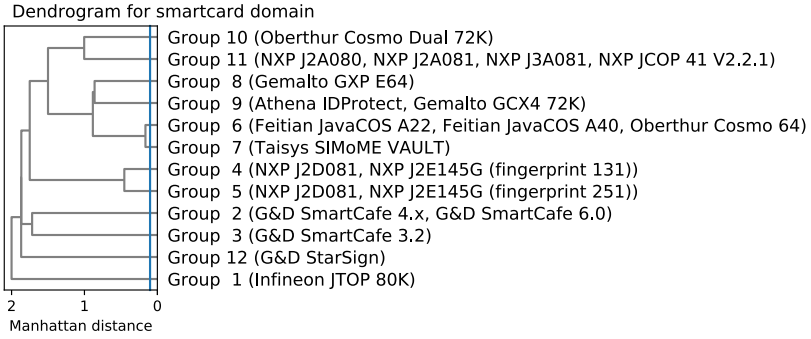

Fig. 2. The clustering of smartcard sources yields 12 separate groups.

4.1 Performance in the Smartcards Domain

The clustering task in the smartcards domain yields 12 recognizable groups for 19 different smartcard models as shown in Fig. 2. The training set for this limited domain contains 20.6 million keys, whereas the test set contains 340 thousand

keys. On average, 61.9% precision and 64.6% recall is achieved. Moreover, 8 out of 12 groups achieve $> 50\%$ precision. Additionally, the classifier exhibits 100% recall on 3 specific groups: a) Infineon smartcards (before 2017 with the ROCA vulnerability [19]), b) G&D Smartcafe 4.x and 6.0, and c) newer G&D Smartcafe 7.0. Figure 3 shows so-called confusion matrix where each row corresponds to percentage of keys in an actual group while each column represents percentage of keys in a predicted group.

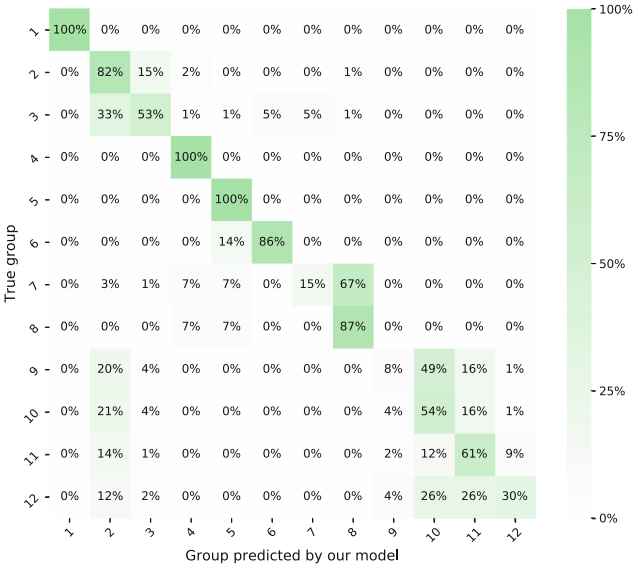

Fig. 3. The confusion matrix for the classifier of a single private key generated in the smartcards domain. A given row corresponds to a vector of observed relative frequencies with which keys generated by a specific group (True group) are misclassified as generated by other groups (Group predicted by our model). For example, group 1 and group 2 have no misclassifications (high accuracy), while keys of group 3 are in 33% cases misclassified as keys from group 2. On average, we achieve 64.6% accuracy. The darker the cell is, the higher number it contains. This holds for all figures in this paper.

As expected, the results represent an improvement when compared to the dataset with all libraries. When one has ten keys of the same card at hand, the expected recall is over 90% on 10 out of 12 groups. The full table of results can be found in the project repository.

Interestingly, 512- and 1024-bit keys generated by the same NXP J2E145G card (similarly also for NXP J2D081) fall into different groups[2]. The main difference is in the modular fingerprint (avoidance of small factors in $p-1$ and $q-1$). We hypothesize that on-card key generation avoids more small factors for

[2] This is an exception to the observation that the selected features behave independently of key length. Otherwise, keys of different length can be used interchangeably.

larger keys. Such behaviour was not observed for other libraries but highlights the necessity of collecting different key lengths in the training dataset when one analyzes black-box proprietary devices or closed-source software libraries.

To summarize, the classification of private keys generated by smartcards is very accurate due to the significant differences resulting from the proprietary, embedded implementations among the different vendors. The differences observed likely results from the requirements to have a smaller footprint required by low-resources devices.

4.2 Performance in the TLS Domain

For the TLS domain, we excluded all the libraries and devices unlikely to be used to generate keys then used by TLS servers. All smartcards are excluded, together with highly outdated or purpose-specific libraries like PGP SDK 4. All hardware security modules (HSMs) are present as they may be used as TLS accelerators or high-security key storage. Summarized, we started with 17 separate cryptographic libraries and HSMs, inspected in a total of 134 versions. The clustering resulted in 13 recognizable groups as shown in Fig. 4.

The domain training set contains 121.8 million keys and the test set contains 1.3 million keys. On average, the classifier achieves 45.5% precision and 42.2% recall. The decrease in average recall compared to the full domain may look surprising, but averaging is deceiving in this context. In fact, recall improved for 10 out of 13 groups that are both in the full set and the TLS domain set, with the precision improving for 9 groups. The mean values of the full dataset are being uplifted by a generally better performance of the model outside the TLS domain. Five groups have >50% precision. OpenSSL (by far the most popular library used by servers for TLS [18]) has 100% recall, making the classification of OpenSSL keys very reliable. Complete results can be found in the project repository.

To summarize, we correctly classify more keys in a more specific TLS domain than with the full dataset classifier. Additionally, the user can be more confident about the decisions of the TLS-specific classifier.

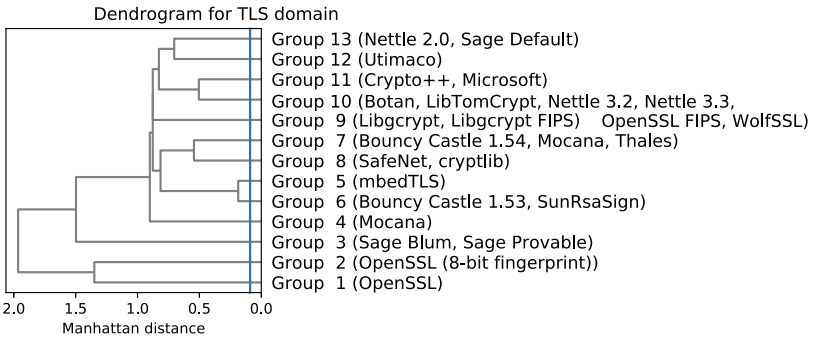

Fig. 4. The clustering of the sources from the TLS domain yields 13 separate groups.

4.3 Performance in the Single-Prime TLS Domain

The rest of this section is motivated by a setting when one wants to analyze a batch of correlated keys. Specifically, we assume a case of $k \geq 1$ keys $(p_1, q_1), \ldots, (p_k, q_k)$ generated by the same source, where $p_1 = p_2 = \cdots = p_k$. This scenario emerges in Sect. 5 and cannot be addressed by previously considered classifiers. If applied, the results would be drastically skewed since the classifier would consider each of p_i separately, putting half of the weight on the shared prime. For that reason, we train a classifier that works on single primes rather than on complete private keys. Instead of feeding the classifier with a batch of k private keys, we supply it with a batch of $k + 1$ unique primes from those keys. The selected features were modified accordingly: we extract the 5 most significant bits from the unique prime, its second least significant bit, and compute the ROCA and modular fingerprint for the single prime. We trained the classifier on the learning set limited to the TLS domain, as in Sect. 4.2.

On average, we achieve 28.8% precision and 36.2% recall when classifying a single prime. Table 3 shows the accuracy results in more detail. It should, however, be stressed that this classifier is meant to be used for batches of many keys at once. When considering a batch of $k \geq 10$ primes, the accuracy is more than 77%. The decrease in accuracy compared to Sect. 4.2 can be explained by the loss of information from the second prime. The features **mod** and **blum** are much less reliable when using only one prime. Since we can compute the most significant bits from a single prime at a time, we lost the information about the ordering of primes (since features **5p** and **5q** are correlated). These facts resulted in only nine separate groups of libraries being distinguishable. The following groups from the TLS domain are no longer mutually distinguishable: 5 and 13, 7 and 11, 8 and 9 and 10.

4.4 Methodology Limitations

The presented methodology has several limitations:

Table 3. Classification accuracy for single-prime features evaluated on TLS domain.

Number of primes in a batch	1	10	20	30	100
Group 1	100.0%	100.0%	100.0%	100.0%	100.0%
Group 2	42.8%	99.7%	100.0%	100.0%	100.0%
Group 3	78.0%	100.0%	100.0%	100.0%	100.0%
Group 4	47.5%	90.3%	95.8%	98.7%	100.0%
Group 5|13	1.8%	30.8%	43.7%	51.8%	74.7%
Group 6	5.2%	48.9%	61.0%	64.8%	76.7%
Group 7|11	0.0%	67.3%	92.3%	97.4%	100.0%
Group 8|9|10	37.9%	99.9%	100.0%	100.0%	100.0%
Group 12	12.8%	61.8%	77.7%	83.9%	97.2%
Average	**36.2%**	77.6%	85.6%	88.5%	94.3%

Classification of an Unseen Source. Not all existing sources of RSA keys are present in our dataset for clustering analysis and classification. This means that attempting to classify a key from a source not considered in our study will bring unpredictable results. The new source may either populate some existing group or have a unique implementation, thus creating a new group. In both cases, the behaviour of the classifier is unpredictable.

Granularity of the Classifier. There are multiple libraries in a single group. The user is therefore not shown the exact source of the key, but the whole group instead. This limitation has two main reasons: 1) Some sources share the same implementation and thus cannot be told apart. 2) The list of utilized features is narrow. There are infinitely many possible features in principle and some may hide valuable information that can further help the model performance. Nevertheless, the proposed methodology allows for an automatic evaluation of features using the naïve Bayes method which shall be considered in future work.

Human Factor. The clustering task in our study requires human knowledge. To be specific, the value of the threshold that splits the libraries into groups (for a particular feature) is established only semi-automatically. We manually confirmed the threshold – when we could explain the difference between the libraries, or moved it otherwise. Summarized, this complicates the fully automatic evaluation on a large number of potential features. Once solved, the relative importance of the individual features could be measured.

5 Real-World GCD-Factorable Keys Origin Investigation

Previous research [2,13,14,16] demonstrated that a non-trivial fraction of RSA keys used on publicly reachable TLS servers is generated insecurely and is practically factorable. This is because the affected network devices were found to independently generate RSA keys that share a single prime or both primes. While an efficient factorization algorithm for RSA moduli is unknown, when two keys accidentally share one prime, the efficient factorization is possible using the Euclidean algorithm to find their GCD[3]. Still, the current number of public keys obtained from crawling TLS servers is too high to allow for the investigation of all possible pairs. However, the distributed GCD algorithm [15] allows analyzing hundreds of millions of keys efficiently. Its performance was sufficient to analyze all keys collected from IPv4-wide TLS scans [5,21] and resulted in almost 1% of factorable keys in the scans collected at the beginning of the year 2016.

After the detection of GCD-factorable keys, the question of their origin naturally followed. Previous research addressed it using two principal approaches: 1) an analysis of the information extractable from the certificates of GCD-factorable keys, and 2) matching specific properties of factored primes with primes generated by a suspected library – OpenSSL. The first approach allowed to detect a range of network routers that seeded their PRNG shortly after boot without

[3] Note that the keys sharing both primes are not susceptible to this attack but reveal their private keys to all other owners of the same RSA key pair.

enough entropy, what caused them to occasionally generate a prime shared with another device. These routers contained a customized version of the OpenSSL library, what was confirmed with the second approach, since OpenSSL code intentionally avoids small factors of $p - 1$ as shown by [17].

While this suite of routers was clearly the primary source of the GCD-factorable keys, are they the sole source of insecure keys? The paper [13] identified 23 router/device vendors that used the code of OpenSSL (using specific OpenSSL fingerprint based on avoidance of small factors in $p - 1$ and information extracted from the certificates). Eight other vendors (DrayRek, Fortinet, Huawei, Juniper, Kronos, Siemens, Xerox, and ZyXEL) produced keys without such OpenSSL fingerprint, and the underlying libraries remained unidentified. In the rest of this section, we build upon the prior work to identify probable sources of the GCD-factorable keys that *do not* originate from the OpenSSL library.

Two assumptions must be met to employ the classifier studied in Sect. 4.3. First, we assume that *when a batch of GCD-factored keys shares a prime, they were all generated by sources from a single classification group.* This conjecture is suggested in [13,14] and supported by the fact that when distinct libraries differ in their prime generation algorithm, they will produce different primes even when initialized from the same seed. On the other hand, when they share the same generation algorithm, they inevitably fall into the same classification group. Second, we assume that *if the malformed keys share only single prime, the PRNG was reseeded with enough entropy before the second prime got generated.* This is suggested by the failure model studied for OpenSSL in [14] and implies that the second prime is generated as it normally would be.

Leveraging these conjectures, the rest of this section tracks the libraries responsible for GCD-factorable keys while not relying on the information in the certificates. First, we describe the dataset gathering process, as well as the factorization of the RSA public keys. Later, successfully factored keys are analyzed, followed with a discussion of findings.

6 Datasets of GCD-Factorable TLS Keys

The input dataset with public RSA keys (both secure and vulnerable ones) was obtained from the Rapid7 archive. All scans between October 2013 and July 2019 (mostly in one or two weeks period) were downloaded and processed, resulting in slightly over 170 million certificates. Only public RSA keys were extracted, and duplicates removed, resulting in 112 million unique moduli. On this dataset, the *fastgcd* [15] tool based on [3] was used to factorize the moduli into private keys. A detailed methodology of this procedure is discussed in Appendix B.

6.1 Batching of GCD-Factorable Keys

Would the precision and recall of our classifier be 100%, one could process the factored keys one by one, establish their origin library and thus detect all sources of insecure keys. But since the classification accuracy of the single-prime TLS

classifier[4] with a single key is only 36%, we apply three adjustments: 1) batch the GCD-factorable keys sharing the same prime (believed to be produced by the same library); 2) analyze only the batches with at least 10 keys (therefore with high expected accuracy); 3) limit the set of the libraries considered for classification only to the single-prime TLS domain. Since the keys from the OpenSSL library were already extensively analyzed by [13], we use the **mod** feature to reliably mark and exclude them from further analysis. By doing so, we concentrate primarily on the non-OpenSSL keys that were not yet attributed. The exact process for classification of factored keys in batches is as follows:

1. Factorize public keys from a target dataset (e.g., Rapid7) using *fastgcd* tool.
2. Form batches of factored keys that share a prime and assume that they originate from the same classification group.
3. Select only the batches with at least k keys (e.g., 10).
4. Separate batches of keys that all carry the OpenSSL fingerprint. As a control experiment, they should classify only to a group with the OpenSSL library.
5. Separate batches without the OpenSSL fingerprint. This cluster contains yet unidentified libraries.
6. Classify the non-OpenSSL cluster using a single-prime TLS classifier.

6.2 Source Libraries Detected in GCD-Factorable TLS Keys

In total, we analyzed more than 82 thousand primes divided into 2511 batches. While each batch has at least 10 keys in it, the median of the batch size is 15. Among the batches, 88.8% of them exhibit the OpenSSL fingerprint. This number well confirms the previous finding by [13] that also captured the OpenSSL-specific fingerprint in a similar fraction of keys. We attribute three other batches as coming from the OpenSSL (8-bit fingerprint), an OpenSSL library compiled to test and avoid divisors of $p-1$ only up to 251. Importantly, slightly more than 11% of batches were generated by some library from groups 8, 9, or 10, which are not mutually distinguishable when only a single prime is available. There are also negative results to report. With the accuracy over 80% (for a batch size of

Table 4. Keys that share a prime factor belong to the same batch. Classification of most batches resulted in OpenSSL as the likely source. The rest of the batches were likely generated by libraries in the combined group 8 | 9 | 10.

Group(s)	# batches
1 (OpenSSL)	2230
2 (8-bit OpenSSL)	3
8 | 9 | 10 (various libraries, see Fig. 4)	278
3; 4; 6; 12; 5 | 13; 7 | 11	0 (*improbable*)

[4] Note that without using single-prime model, the results are biased as the shared prime is considered multiple times in the classification process.

15) and no batches attributed to any of groups 3, 4, 6, 12, 5 | 13, or 7 | 11, it is very improbable that any GCD-factorable keys originate from the respective sources in these libraries (Table 4).

7 Related Work

The fingerprinting of devices based on their physical characteristics, exposed interfaces, behaviour in non-standard or undefined situations, errors returned, and a wide range of various other side-channels is a well-researched area. The experience shows that finding a case of a non-standard behaviour is usually possible, while making a group of devices indistinguishable is very difficult due to an almost infinite number of observable characteristics, resulting in an arms race between the device manufacturers and fingerprinting observers.

Having the device fingerprinted is helpful to better understand the complex ecosystem like quantifying the presence of interception middle-boxes on the internet [9], types of clients connected or version of the operating system. Differences may help point out subverted supply chains or counterfeit products.

When applied to the study of cryptographic keys and cryptographic libraries, researchers devised a range of techniques to analyze the fraction of encrypted connections, the prevalence of particular cryptographic algorithms, the chosen key lengths or cipher suites [1,2,4,8,10,12,24]. Information about a particular key is frequently obtained from the metadata of its certificate.

Periodical network scans allow to assess the impact of security flaws in practice. The population of OpenSSL servers with the Heartbleed vulnerability was measured and monitored by [7], and real attempts to exploit the bug were surveyed. If the necessary information is coincidentally collected and archived, even a backward introspection of a vulnerability in time might be possible.

The simple test for the ROCA vulnerability in public RSA keys allowed to measure the fraction of citizens of Estonia who held an electronic ID supported by a vulnerable smartcard, by inspecting the public repository of eID certificates [19]. The fingerprinting of keys from smartcards was used to detect that private keys were generated outside of the card and injected later into the eIDs, despite the requirement to have all keys generated on-card [20].

The attribution of the public RSA key to its origin library was analyzed by [23]. Measurements on large datasets were presented in [18], leading to accurate estimation of the fraction of cryptographic libraries used in large datasets like IPv4-wide TLS. While both [23] and [18] analyze the public keys, private keys can be also obtained under certain conditions of faulty random number generator [6,13,14,16,22]. The origin of weak factorable keys needs to be identified in order to notify the maintainers of the code to fix underlying issues. A combination of key properties and values from certificates was used.

8 Conclusions

We provide what we believe is the first wide examination of properties of RSA keys with the goal of attribution of private key to its origin library.

The attribution is applicable in multiple scenarios, e.g., to the analysis of GCD-factorable keys in the TLS domain. We investigated the properties of keys as generated by 70 cryptographic libraries, identified biased features in the primes produced, and compared three models based on Bayes classifiers for the private key attribution.

The information available in private keys significantly increases the classification performance compared to the result achieved on public keys [23]. Our work enables to distinguish 26 groups of sources (compared to 13 on public keys) while increasing the accuracy more than twice w.r.t. random guessing. When 100 keys are available for the classification, the correct result is almost always provided (>99%) for 19 out of 26 groups.

Finally, we designed a method usable also for a dataset of keys where one prime is significantly correlated. Such primes are found in GCD-factorable TLS keys where one prime was generated with insufficient randomness and would introduce a high classification error in the unmodified method. As a result, we can identify libraries responsible for the production of these GCD-factorable keys, showing that only three groups are a relevant source of such keys. The accurate classification can be easily incorporated in forensic and audit tools.

While the bias in the keys usually does not help with factorization, the cryptographic libraries should approach their key generation design with a great care, as strong bias can lead to weak keys [19]. We recommend to follow a key generation process with as little bias present as possible.

Acknowledgements. The authors would like to thank anonymous reviewers for their helpful comments. P. Svenda and V. Matyas were supported by Czech Science Foundation project GA20-03426S. Some of the tools used and other people involved were supported by the CyberSec4Europe Competence Network. Computational resources were supplied by the project e-INFRA LM2018140. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

A Classifier Results Discussion and Datasets Preparation

Some groups are accurately classified and rarely misclassified even with a single key available: namely group 1 (Infineon prior 2017, distinct because of the ROCA fingerprint), group 2 (Giesecke&Devrient SmartCafe 4.x and 6.0), group 24 (standard OpenSSL *without* the FIPS module enabled) and group 26 (Giesecke&Devrient SmartCafe 7.0) are all classified with more than 96% recall. Groups 1, 2, and 26 are rarely misclassified as origin library (false positive). The keys from group 25 (OpenSSL avoiding only 8-bit small factors in $p-1$) are misclassified as group 24 (standard OpenSSL) in 31.6% cases, which still identifies the origin library correctly, only misidentifies the OpenSSL compile-time configuration.

In contrast, keys from groups 7, 10, 11, 14, 15, and 17 are almost always misclassified (less than 8% recall, some even less than 1%). However, as dis-

Table 5. The average classification accuracy of the best performing Bayes classifier. In the i-th column we consider a classifier successful if the true source of the key is among i best guesses of our model. Similarly, for each of the 3 columns we evaluate the success rate when $1, 2, 3, 5$ or 10 keys from the same group are available.

	Top 1 match					Top 2 match					Top 3 match				
#keys in batch	1	2	3	5	10	1	2	3	5	10	1	2	3	5	10
Group 1	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Group 2	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Group 3	86.3%	98.1%	99.8%	100.0%	100.0%	98.2%	100.0%	100.0%	100.0%	100.0%	98.2%	100.0%	100.0%	100.0%	100.0%
Group 4	92.7%	99.3%	99.9%	100.0%	100.0%	94.8%	99.7%	100.0%	100.0%	100.0%	96.4%	99.9%	100.0%	100.0%	100.0%
Group 5	60.8%	76.3%	79.8%	90.7%	96.6%	71.5%	90.1%	93.6%	98.7%	99.9%	73.0%	91.3%	97.6%	98.8%	100.0%
Group 6	73.0%	88.1%	88.5%	83.5%	69.8%	92.8%	92.8%	97.7%	98.2%	99.9%	96.5%	97.0%	99.5%	99.8%	100.0%
Group 7	7.6%	18.9%	30.0%	47.9%	73.6%	77.3%	95.5%	98.8%	99.9%	100.0%	92.7%	99.3%	99.9%	100.0%	100.0%
Group 8	16.3%	33.5%	44.2%	54.6%	62.8%	27.5%	56.2%	73.5%	91.3%	99.2%	38.6%	63.9%	81.7%	94.2%	99.5%
Group 9	12.8%	28.3%	38.9%	50.9%	61.1%	37.7%	65.7%	79.1%	90.4%	99.0%	48.3%	75.9%	87.8%	96.8%	99.8%
Group 10	0.0%	24.7%	47.7%	67.9%	92.0%	18.4%	44.1%	60.8%	79.8%	96.1%	52.7%	87.6%	92.5%	98.5%	100.0%
Group 11	6.9%	21.8%	34.2%	51.6%	63.1%	56.7%	87.2%	95.9%	99.4%	100.0%	73.2%	95.2%	99.2%	100.0%	100.0%
Group 12	54.9%	75.4%	78.2%	71.5%	65.8%	72.2%	85.0%	95.4%	98.1%	100.0%	89.5%	95.7%	99.0%	99.8%	100.0%
Group 13	47.2%	57.0%	69.6%	84.8%	96.3%	52.9%	68.6%	80.9%	93.8%	99.5%	66.4%	82.9%	91.4%	98.0%	99.8%
Group 14	6.9%	22.4%	40.8%	70.5%	93.6%	7.7%	41.0%	69.7%	90.8%	99.3%	12.4%	53.7%	78.9%	95.4%	99.9%
Group 15	0.2%	28.0%	52.7%	80.0%	96.5%	2.5%	43.4%	65.4%	90.2%	99.4%	28.2%	64.6%	81.0%	94.4%	99.7%
Group 16	31.4%	63.6%	79.4%	91.1%	99.4%	40.9%	70.6%	85.4%	96.5%	100.0%	48.3%	80.0%	92.1%	98.8%	100.0%
Group 17	5.1%	28.6%	50.2%	78.0%	97.6%	18.3%	51.2%	71.9%	92.0%	99.7%	37.7%	73.0%	89.0%	98.1%	100.0%
Group 18	12.2%	55.1%	70.5%	78.5%	84.7%	45.2%	91.0%	98.2%	100.0%	100.0%	76.3%	96.1%	99.4%	100.0%	100.0%
Group 19	44.0%	54.4%	59.7%	67.3%	78.5%	54.5%	88.3%	97.3%	99.9%	100.0%	62.1%	93.8%	99.1%	100.0%	100.0%
Group 20	81.5%	95.2%	98.7%	99.9%	100.0%	97.2%	100.0%	100.0%	100.0%	100.0%	98.9%	100.0%	100.0%	100.0%	100.0%
Group 21	53.0%	77.9%	88.4%	97.0%	99.9%	95.2%	99.7%	100.0%	100.0%	100.0%	97.6%	100.0%	100.0%	100.0%	100.0%
Group 22	14.6%	39.2%	53.5%	72.5%	92.3%	78.0%	98.2%	99.8%	100.0%	100.0%	97.2%	99.9%	100.0%	100.0%	100.0%
Group 23	77.4%	98.0%	99.9%	100.0%	100.0%	96.8%	99.9%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Group 24	96.8%	99.9%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Group 25	58.3%	86.7%	96.1%	99.7%	100.0%	87.6%	97.9%	99.6%	100.0%	100.0%	93.9%	99.7%	100.0%	100.0%	100.0%
Group 26	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Average	**47.7%**	64.2%	73.1%	82.2%	89.4%	**66.3%**	83.3%	90.9%	96.9%	99.7%	**76.1%**	90.4%	95.7%	98.9%	100.0%

cussed in the next section, if some additional information is available and can be considered, this misclassification can be largely remediated.

Keys from group 7 (Libgcrypt) are mostly misclassified as group 6 (PGP SDK 4, 64.5%) or group 13 (Gemalto GXP E64, 20.2%). As libgcrypt is a commonly used library while groups 6 and 13 correspond to a very old library and card, this case demonstrates the possibility for further classifier improvement when some prior knowledge is available. E.g., for the TLS domain, groups corresponding to old smartcards or non-TLS libraries can be ruled out from the process.

Group 10 (Bouncy Castle since 1.54, Mocana 7.x or HSM Thales nShieldF3) is misclassified as group 12 (smartcard Taisys SIMoME, 36.3%) or group 5 (Mocana 6.x 21.0%). Additional information can improve classification accuracy as the Taisys smartcard is unlikely source for the most usage domains. If Mocana library actually generated the key, only the identified version is incorrect.

Group 11 (cryptlib, Safenet HSM Luna SA-1700, and Feitian and Oberthur cards) is misclassified as group 12 (smartcard Taisys, 50.2%) or group 20 (Oberthur Cosmo Dual, 20.4%). This is a very similar case as for group 10.

Group 14 (Microsoft and Crypto++, prevalent group) is misclassified as group 6 (PGP SDK 4, 23.9%), group 12 (card Taisys, 20.1%), group 13 (card Gemalto GXP E64, 13.5%) or group 5 (Mocana 6.x, 10.7%). Again, for the TLS domain, the only real misclassification problem is with the Mocana 6.x library.

Group 15 (large group with multiple frequently used libraries) is misclassified as group 12 (card Taisys, 27.2%), group 13 (card Gemalto GXP E64, 18.1%),

group 20 (card Oberthur, 11.7%) or group 6 (PGP SDK 4, 32.3%). For the TLS domain, no group from the misclassified ones is likely.

Group 17 (Nettle, Cryptix, FlexiProvider) is misclassified as multiple other groups where only groups 5 (Mocana 6.x) and 9 (Bouncy Castle prior 1.54 and SunRsaSign OpenJDK 1.8) cannot be ruled out as unlikely for the TLS domain (Table 5).

B Obtaining Dataset of GCD-Factorable Keys

The *fastgcd* [15] tool based on [3] was used to perform the search for the GCD-factorable keys. Only valid RSA keys were considered[5]. Running the *fastgcd* tool for a high number of keys (around 112 million for Rapid7 dataset) requires an extensive amount of RAM. Running the tool on a machine with 500 GB of RAM resulted in only a few factored keys, all sharing just tiny factors, while the tool did not produce any errors or warnings. The same computation on a subset of 10 million keys revealed a substantial number of large factors. Likely, the *fastgcd* tool requires even more RAM for the correct functioning with such a large number of keys. To solve the problem, we partitioned the time-ordered dataset into two subsets of 50 and 62 million keys with an additional third subset with 50 million keys that partially overlapped both previous partitions. By doing so, we miss GCD-factorable keys that appeared in the dataset separated by a considerable time distance (2–3 years). We hypothesise that if a prevalent source starts producing GCD-factorable keys, we capture a sufficiently large batch of them within a single subset. In total, we have acquired 114 thousand unique factors from the whole dataset.

References

1. Albrecht, M.R., Degabriele, J.P., Hansen, T.B., Paterson, K.G.: A surfeit of SSH cipher suites. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1480–1491. ACM (2016)
2. Barbulescu, M., Stratulat, A., Traista-Popescu, V., Simion, E.: RSA weak public keys available on the internet. In: Bica, I., Reyhanitabar, R. (eds.) SECITC 2016. LNCS, vol. 10006, pp. 92–102. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47238-6_6
3. Bernstein, D.J.: How to find smooth parts of integers (2004). [cit. 2020-07-13]. http://cr.yp.to/papers.html#smoothpart
4. Cangialosi, F., et al.: Measurement and analysis of private key sharing in the https ecosystem. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 628–640. ACM (2016)
5. Censys: Censys TLS Full IPv4 443 Scan (2015). [cit. 2020-07-13]. https://censys.io/data/443-https-tls-full_ipv4/historical
6. Batch-GCDing Github SSH Keys (2015). [cit. 2020-07-13]. https://cryptosense.com/batch-gcding-github-ssh-keys/

[5] The factorization occasionally finds small prime factors up to 2^{16}, likely because the public key (certificate) was damaged, e.g., by a bit flip.

7. Durumeric, Z., et al.: The matter of heartbleed. In: Proceedings of the 2014 Conference on Internet Measurement Conference, pp. 475–488. ACM (2014)
8. Durumeric, Z., Kasten, J., Bailey, M., Halderman, J.A.: Analysis of the HTTPS certificate ecosystem. In: Proceedings of the 2013 ACM Internet Measurement Conference, pp. 291–304. ACM (2013)
9. Durumeric, Z., et al.: The security impact of https interception. In: Network and Distributed Systems Symposium. The Internet Society (2017)
10. Electronic Frontier Foundation: The EFF SSL Observatory (2010). [cit. 2020–07-13]. https://www.eff.org/observatory
11. Flach, P.: Machine Learning: The Art and Science of Algorithms that Make Sense of Data, Chap. 2, pp. 57–58. Camridge University Press (2012)
12. Gustafsson, J., Overier, G., Arlitt, M., Carlsson, N.: A first look at the CT landscape: certificate transparency logs in practice. In: Kaafar, M.A., Uhlig, S., Amann, J. (eds.) PAM 2017. LNCS, vol. 10176, pp. 87–99. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54328-4_7
13. Hastings, M., Fried, J., Heninger, N.: Weak keys remain widespread in network devices. In: Proceedings of the 2016 ACM on Internet Measurement Conference, pp. 49–63. ACM (2016)
14. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your Ps and Qs: detection of widespread weak keys in network devices. In: Proceeding of USENIX Security Symposium, pp. 205–220. USENIX (2012)
15. Heninger, N., Halderman, J.A.: Fastgcd (2015). [cit. 2020-07-13]. https://github.com/sagi/fastgcd
16. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Ron was wrong, whit is right. Cryptology ePrint Archive, Report 2012/064 (2012). [cit. 2020-07-13]. https://eprint.iacr.org/2012/064
17. Mironov, I.: Factoring RSA Moduli II (2012). [cit. 2020–07-13]. https://windowsontheory.org/2012/05/17/factoring-rsa-moduli-part-ii/
18. Nemec, M., Klinec, D., Svenda, P., Sekan, P., Matyas, V.: Measuring popularity of cryptographic libraries in internet-wide scans. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 162–175. ACM (2017)
19. Nemec, M., Sys, M., Svenda, P., Klinec, D., Matyas, V.: The return of coppersmith's attack: practical factorization of widely used RSA Moduli. In: 24th ACM Conference on Computer and Communications Security (CCS 2017), pp. 1631–1648. ACM (2017)
20. Parsovs, A.: Estonian electronic identity card: security flaws in key management. In: 29th USENIX Security Symposium. USENIX Association (2020)
21. Rapid7: Rapid 7 Sonar SSL full IPv4 scan (2019). [cit. 2020-07-13]. https://opendata.rapid7.com/sonar.ssl/
22. Software in the Public Interest: DSA-1571-1 openssl - predictable random number generator (2008). [cit. 2020-07-13]. https://www.debian.org/security/2008/dsa-1571
23. Svenda, P., et al.: The million-key question—investigating the origins of RSA public keys. In: Proceeding of USENIX Security Symposium, pp. 893–910 (2016)
24. VanderSloot, B., Amann, J., Bernhard, M., Durumeric, Z., Bailey, M., Halderman, J.A.: Towards a complete view of the certificate ecosystem. In: Proceedings of the 2016 ACM on Internet Measurement Conference, pp. 543–549. ACM (2016)