Measuring Popularity of Cryptographic Libraries in Internet-Wide Scans

Matus Nemec
Masaryk University,
Ca' Foscari University of Venice
mnemec@mail.muni.cz

Dusan Klinec
EnigmaBridge, Masaryk University
dusan@enigmabridge.com

Petr Svenda
Masaryk University
svenda@fi.muni.cz

Peter Sekan
Masaryk University
peter.sekan@mail.muni.cz

Vashek Matyas
Masaryk University
matyas@fi.muni.cz

ABSTRACT

We measure the popularity of cryptographic libraries in large datasets of RSA public keys. We do so by improving a recently proposed method based on biases introduced by alternative implementations of prime selection in different cryptographic libraries. We extend the previous work by applying statistical inference to approximate a share of libraries matching an observed distribution of RSA keys in an inspected dataset (e.g., Internet-wide scan of TLS handshakes). The sensitivity of our method is sufficient to detect transient events such as a periodic insertion of keys from a specific library into Certificate Transparency logs and inconsistencies in archived datasets.

We apply the method on keys from multiple Internet-wide scans collected in years 2010 through 2017, on Certificate Transparency logs and on separate datasets for PGP keys and SSH keys[1]. The results quantify a strong dominance of OpenSSL with more than 84% TLS keys for Alexa 1M domains, steadily increasing since the first measurement. OpenSSL is even more popular for GitHub client-side SSH keys, with a share larger than 96%. Surprisingly, new certificates inserted in Certificate Transparency logs on certain days contain more than 20% keys most likely originating from Java libraries, while TLS scans contain less than 5% of such keys.

Since the ground truth is not known, we compared our measurements with other estimates and simulated different scenarios to evaluate the accuracy of our method. To our best knowledge, this is the first accurate measurement of the popularity of cryptographic libraries not based on proxy information like web server fingerprinting, but directly on the number of observed unique keys.

KEYWORDS

RSA algorithm, cryptographic library, prime generation

[1]Full details, processing scripts, datasets and supplementary materials can be found at https://crocs.fi.muni.cz/papers/acsac2017.

ACSAC 2017, December 4–8, 2017, San Juan, PR, USA

ACM ISBN 978-1-4503-5345-8/17/12…$15.00
https://doi.org/10.1145/3134600.3134612

1 INTRODUCTION

With solid mathematical foundations for the currently used cryptographic algorithms like RSA or AES, a successful attack (a compromise of used keys or exchanged messages, a forgery of signatures, etc.) is achieved only very infrequently through mathematical breakthroughs, but dominantly by a compromise of secrets at the end-points, by attacks on the protocol level or via so-called implementation attacks, often combined with an immense computational effort required from the attacker.

Implementation attacks exploit some shortcomings or a specific behavior of the software leading to unintended data leakages in otherwise mathematically secure algorithms. A large number of practical attacks in recent years [31] testifies how difficult it is to make an implementation secure, robust and without side-channel leakage. Even major libraries such as OpenSSL, Java JCE or Microsoft CryptoAPI were hit by multiple problems including extraction of RSA private keys [11] or AES secret keys [10] remotely from a targeted web server and generation of vulnerable keys by a weak or a malfunctioning random generator [1, 24]. It is reasonable to expect that similar problems will occur in future for these and other cryptographic libraries as well.

The prediction of an impact for a future bug depends not only on the nature of the bug (unknown in advance) but also on the overall popularity of the affected cryptographic library within the targeted usage domain. A security bug in OpenSSL will probably cause more harm than a bug in an unknown or sparsely used library.

Yet the estimation of the popularity of a given library is a complicated affair. As a library produces random keys, it is difficult to attribute a particular key to its originating library based only on the bits of the key. A common approach is to make indirect estimates based on additional information such as specific strings inserted into certificates, default libraries used by a software package which is identified by other means (e.g., the Apache HTTP Server typically uses OpenSSL) or specific key properties (uncommon key lengths or domain parameters). All these approaches leave a large uncertainty about the real origin of the target key. A certificate can be crafted by a different software than its key was, a server key may be imported, and a combination of an algorithm and key length are only rarely specific to a single library.

Our work aims to accurately measure the popularity of libraries based on the subtle biases in bits of RSA public keys due to different implementations of the prime pair selection process, as recently described in [36]. The bias has almost no impact on the entropy

of a key and poses no threat with respect to factorization attacks. However, it allows for a probabilistic attribution of a key to the originating library. We focus on answering the following questions:

(1) *How many keys in an inspected dataset originate from specific cryptographic libraries?*
(2) *How does the popularity of cryptographic libraries change over time? Can we detect sudden temporary changes?*
(3) *What library generated a single given RSA key if the key usage domain (TLS, SSH, etc.) is known?*

In the original work, all libraries were assigned the same prior probability – an assumption that is certainly inaccurate (intuitively, OpenSSL is a far more common source of TLS keys than PGP software). We propose an improved method that automatically extracts the prior probability directly from a large dataset – obtaining the popularity of libraries in the inspected dataset and subsequently improving the classification accuracy of individual keys.

The answer to the first question tells us the popularity of cryptographic libraries in different usage domains. Since the method is based on the actual key counts instead of anecdotal proxies (e.g., installed packages or server strings), it is significantly more accurate. Besides providing usage statistics, the popularity of the libraries is important when estimating the potential impact of (past and future) critical flaws, as well as when deciding where to most efficiently spend the effort on development and security code review.

The availability of large Internet-wide scans of TLS handshakes performed every week, Certificate Transparency logs and append-only PGP keyserver databases, allow us to perform a study of cryptographic library popularity over time, hence to find an answer to the second question. When the scans are performed as frequently as every week or even every day, temporary changes in the popularity ratio can reveal sudden changes in the distributions of the keys, possibly making a library more prominent than expected. Such phenomena may indicate users reacting to a disclosed vulnerability (e.g., by replacing their keys) or some significant changes in security procedures of server implementations.

Finally, an accurate answer to the third question allows us to reveal the originating library of a particular key. The previous work [36] correctly labeled the origin of about 40% of random keys, when a single public key was classified in a simulation with evenly probable libraries. We improved the accuracy to *over 94%* for prior probabilities of libraries typical for the TLS domain.

Contributions. Our paper brings the following contributions:

- A method for an accurate survey of popularity of cryptographic libraries based on matching observed counts of RSA keys to a mixture of biased reference distributions produced by the libraries.
- Analyses of usage trends for large real-world archived datasets of certificates for TLS, SSH and PGP from 2010 through 2017.
- Detection and analysis of abrupt transient events manifested by a sudden change in the ratio of libraries.
- Release of the classification tool and extensible catalog of more than 60 profiles for open/closed-source software libraries, hardware security modules, cryptographic smartcards and tokens.

The rest of the paper is organized as follows: Section 2 provides the necessary background for understanding the RSA key classification method based on slight biases in the distribution of keys and a basic overview of the automatic extraction of prior probabilities from an inspected dataset. Section 3 explains the details of the library popularity measurement method and discusses the accuracy. Section 4 applies our method to large current and archived datasets to measure the popularity of libraries in time and discusses the observed results. Section 5 provides a review of related work. The paper is concluded in Section 6.

2 METHOD OVERVIEW

The authors of [36] demonstrated how different implementation choices made by developers of cryptographic libraries lead to biases in generated RSA keys. To generate an RSA key pair, two large random primes p and q (typically half of the binary length of the modulus) must be found. The modulus N is the product of the primes. None of the sources of keys examined by [36] produced moduli with uniformly distributed most significant byte, as one might expect from cryptographic keys. Instead, the distributions of moduli of each source were determined by the choice of the primes.

In order to reduce the uncertainty about the origin of a particular key, three conditions must be satisfied: 1) bias is present in the key, 2) reference distributions of (ideally) all implementations are known, and 3) a suitable method exists to match the reference data and the observed data.

We use the same biases as observed in [36]. We collected reference distributions from additional sources and other versions of cryptographic libraries, extending the knowledge of possible key origins. The original classification method was based on conditional probabilities and the application of Bayes' rule. The origin (a group of sources) of a key could be correctly estimated in 40% of attempts – as opposed to 7.7% success of a random guess. We devised a new method that estimates the proportion of sources in a given dataset and more than doubles the average accuracy in TLS datasets.

2.1 Choice of key features

The most common reasons for the biases in the private primes were efficiency improvements, a special form of the primes, bugs, and uncommon implementation choices. The biases propagate to public moduli to a certain degree – some are directly observable, some require a large number of keys to distinguish and some cannot be seen from the public values. This calls for a creation of a mask of the public keys – instead of dealing with the full keys, some properties are extracted and each key is represented by a vector of features. We use the following features, inspired by the original approach:

(1) *The most significant bits of the modulus (2^{nd} to 7^{th} bit):* The highest bits of the primes are often set to a constant, e.g., the two highest bits set to 1 to ensure the modulus has the correct bit length. The high bits are sometimes manipulated further, up to four bits were determined non-randomly. Even without directly manipulating the top bits, the intervals from which the primes are chosen are seen in the top bits of the modulus.
(2) *The modulus modulo 4:* Due to bugs and unusual (for RSA) implementation choices, the moduli might end up being Blum integers – due to the primes always being equal to 3 modulo 4, the moduli are always equal to 1 modulo 4.

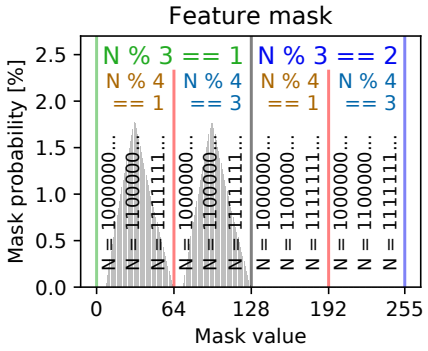

Figure 1: Features extracted from a public RSA modulus. The moduli generated by OpenSSL are always equal to one modulo three ($N \% 3 = 1$), but they are uniformly distributed modulo four ($N \% 4 = 1$ and $N \% 4 = 3$, with the same probability). The most significant bits of the moduli are never equal to 1000_2, and have the value 1100_2 more frequently than 1111_2.

(3) *The modulus modulo 3:* Another unexplained implementation decision (in OpenSSL and elsewhere) avoids primes p if $p - 1$ has small divisors, other than 2. If $p - 1$ and $q - 1$ are never divisible by 3, then the modulus is always equal to 1 modulo 3 and never equal to 2 modulo 3. For larger prime divisors (5, 7, 11, etc.), the property is not directly observable from a single modulus and is therefore impractical for key classification.

We rely on the deep analysis of the key generation process and statistical properties of the resulting keys presented in [36]. We construct our mask similarly, however, we decided to drop the feature encoding the bit length of the modulus – as it is only relevant for one uncommon library implementation.

Our choice of the mask is reflected in Figure 1, illustrated by the distributions of mask values for OpenSSL. When compared to [36], we also changed the order of the features in the mask, to allow for an easier interpretation of distributions from the illustrations. The relevant biases for all sources are listed in Table 3 in the Appendix.

2.2 Clustering analysis

For each source we generate a large number of keys (typically one million), extract the features from the keys according to the feature mask and compute the distributions of the feature masks. The previous research [36] used a highly representative sample of cryptographic libraries in what were then the most recent available versions. For this research, we also added keys from two hardware security module devices, two cryptographic tokens and from the PuTTy software, a popular SSH implementation for Microsoft Windows. The latter is valuable when we consider the domain of SSH authentication keys collected from GitHub.

We also collected new keys from the latest implementations of the considered libraries, this unveiled a change in the behavior of Libgcrypt 1.7.6 in the FIPS mode. Furthermore, we added earlier releases of several libraries, to support the claims made about older datasets. However, we only detected a change in the algorithm for Nettle 2.0, when comparing the libraries with their current versions. Most notably, the addition of new sources did not change the results of clustering analysis as performed previously – the number of groups and their division remains mostly unchanged.

Since we examine many libraries across several versions, we often encounter very similar distributions (i.e., the algorithm did not change across versions or multiple libraries use the same algorithm). Since these distributions are not mutually distinguishable, we use clustering analysis to create clusters (groups) of sources with very similar properties. We use the Euclidean distance as the metric with a bound (threshold) for the creation of clusters, exactly as applied in [36]. The result of the clustering analysis is visualized in Figure 2 as a dendrogram. Instead of working with individual libraries, we do the analysis with the groups. Even though we cannot differentiate between libraries inside a group, luckily, the most popular libraries (OpenSSL, Microsoft) are represented by a distinct or a very small group.

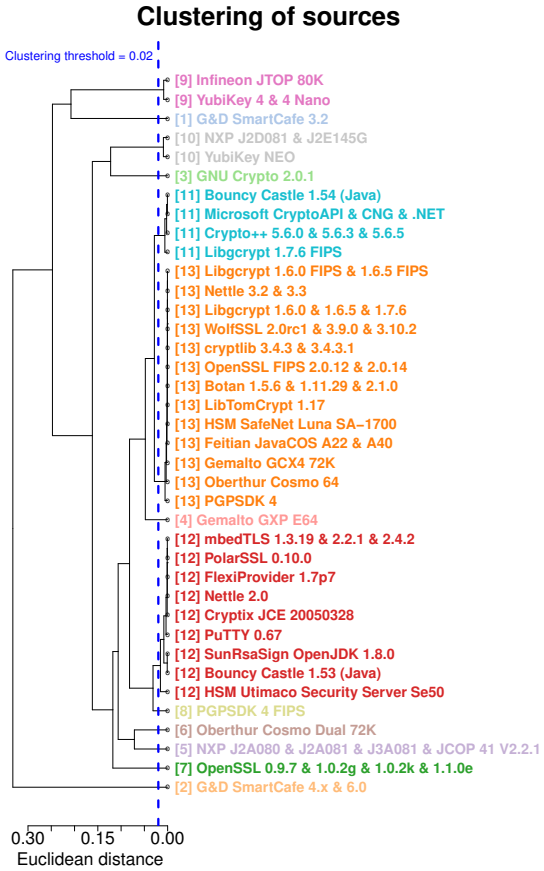

Figure 2: The result of the clustering analysis visualized as a dendrogram. Clusters are created based on the Euclidean distance, with a separation threshold of 0.02 (blue dashed line), similarly as the approach of [36]. The group numbers are listed in brackets next to the source name.

2.3 Dataset classification – original approach

The main focus of the authors of [36] was to get information about the origin (the most probable group G) of a particular key K. To achieve it, the authors applied the Bayes' rule:

$$P(G|K) = \frac{P(K|G)P(G)}{P(K)}, \tag{1}$$

where $P(G|K)$ is the conditional probability that a group G was used to generate a given key K (the aim of the classification), $P(K|G)$ is the conditional probability that a key K is generated by a given group G (obtained from the reference distributions), $P(G)$ is the prior probability of a group G and $P(K)$ is the probability of a key K in a dataset. The highest numerical value of $P(G|K)$ corresponds to the first guess on the most probable group G.

To reason about the popularity of libraries in large datasets, all keys were considered separately and then the information was summarized. Among the main shortcomings of the method was the assumption that cryptographic libraries (alternatively, groups of libraries) are chosen evenly by users (i.e., the prior probability $P(G)$ is equal for all groups G), which is evidently false. The method also failed to consider the "big picture" – keys were considered in small batches (e.g., a single key or a few keys assumed to originate from a same source), hence the probability $P(K)$ of a key was usually 1.

2.4 Dataset classification – our approach

We improved the method in the following way: to estimate the origin of a key, we use an appropriate prior probability $P(G)$ for the domain where the key can be found (e.g., different for TLS, PGP, SSH...). To our best knowledge, no reliable estimates of the prior probability $P(G)$ were published for large domains. We therefore propose and apply our own method for estimating the proportion of cryptographic libraries in large datasets, based on statistical inference. In this way, we construct a tailored prior probability estimate from the "big picture" before we make claims about individual keys.

To accomplish the prior probability estimation, we create a model based on our reference distributions and we search for parameters of the model that best match the whole observed sample. We use a numerical method – the non-negative least squares fit (NNLSF) [30]. It is the standard least squares fit method with a restriction on the parameters, since the probabilities must be non-negative. A detailed description of the methodology is given in Section 3.

The described approach also provides more than a two-fold increase in the accuracy of origin estimation for public keys when compared to the original approach of [36] in the domain of TLS. The improvement is due to the application of the obtained prior probabilities. More details and accuracy measurements for the prior probability estimation itself are discussed in Section 3.4.

2.5 Limitations

Individual sources that belong to a single group cannot be mutually distinguished. Fortunately, the two most significant TLS libraries belong to small groups – OpenSSL is the single source in Group 7 and Microsoft libraries share Group 11 only with Crypto++ and two recently introduced library versions – Bouncy Castle since version 1.54 (December 2015) and Libgcrypt 1.7.6 FIPS (January 2017).

The particular version of a library cannot be identified, only a range of versions with the same key generation algorithm. E.g., Bouncy Castle from version 1.53 can be differentiated from version 1.54 due to a change in key generation, but not from version 1.52 that shares the same code in the relevant methods.

Based on our simulations, an accurate prior probability estimation requires a dataset with at least 10^5 keys. However, note that the classification of a single key is still possible and on average benefits greatly from the accurate prior probability of its usage domain.

3 METHODOLOGY IN DETAIL

When aiming to estimate the library usage in a given domain, we create a model of the domain backed by reference distributions collected from known sources. We obtain RSA keys from the target domain and search for the parameters of our model which fit the observed data. We use the model and the estimated library probabilities to classify individual keys according to their origin.

3.1 Model

We assume there are m groups of sources, created by clustering analysis (Section 2.2) based on the similarity of the distributions of generated public keys. The probability $P(K)$ that a randomly chosen key in the sample has a particular mask value K (the feature mask is explained in Section 2.1) is given by:

$$P(K) = \sum_{j=1}^{m} P(G_j)P(K|G_j), \tag{2}$$

which is the sum of probabilities $P(G_j)P(K|G_j)$ over all m groups G_j, where $P(G_j)$ is the probability that a source from a group G_j is chosen in a particular domain (*prior probability of source in the domain*) and $P(K|G_j)$ is the conditional probability of generating a key with mask K when a library from the group G_j is used.

The probabilities $P(K|G_j)$ are estimated by generating a large number of keys from available sources, which represent the reference distributions of the key masks (*the profile of the group*). The probability of a key K in a dataset of real-world keys is approximated as $\#K/D$, where $\#K$ is the number of keys with the mask K and D is the number of all keys in the dataset.

3.2 Prior probability estimation

The process of estimating prior probability is completely automated and does not require any user input. This fact allows us to construct an independent estimate of library usage from public keys only, without an influence of other information.

We find what are the likely prior probabilities of libraries that would lead to the observed distribution of keys in a given sample, based on the reference group profiles. The principle is illustrated in Figure 3 – the observed distribution is reconstructed by combining the 13 distributions in a specific ratio (prior probability estimated by our approach). Intuitively, for a good estimate of prior probabilities it is necessary (but not always sufficient) that the observed and the reconstructed distributions match closely.

For each of n possible values of mask K, we substitute observed values into Equation 2. In our case, the system has 256 equations with 13 unknowns $P(G_j)$. Since both the distribution of real world

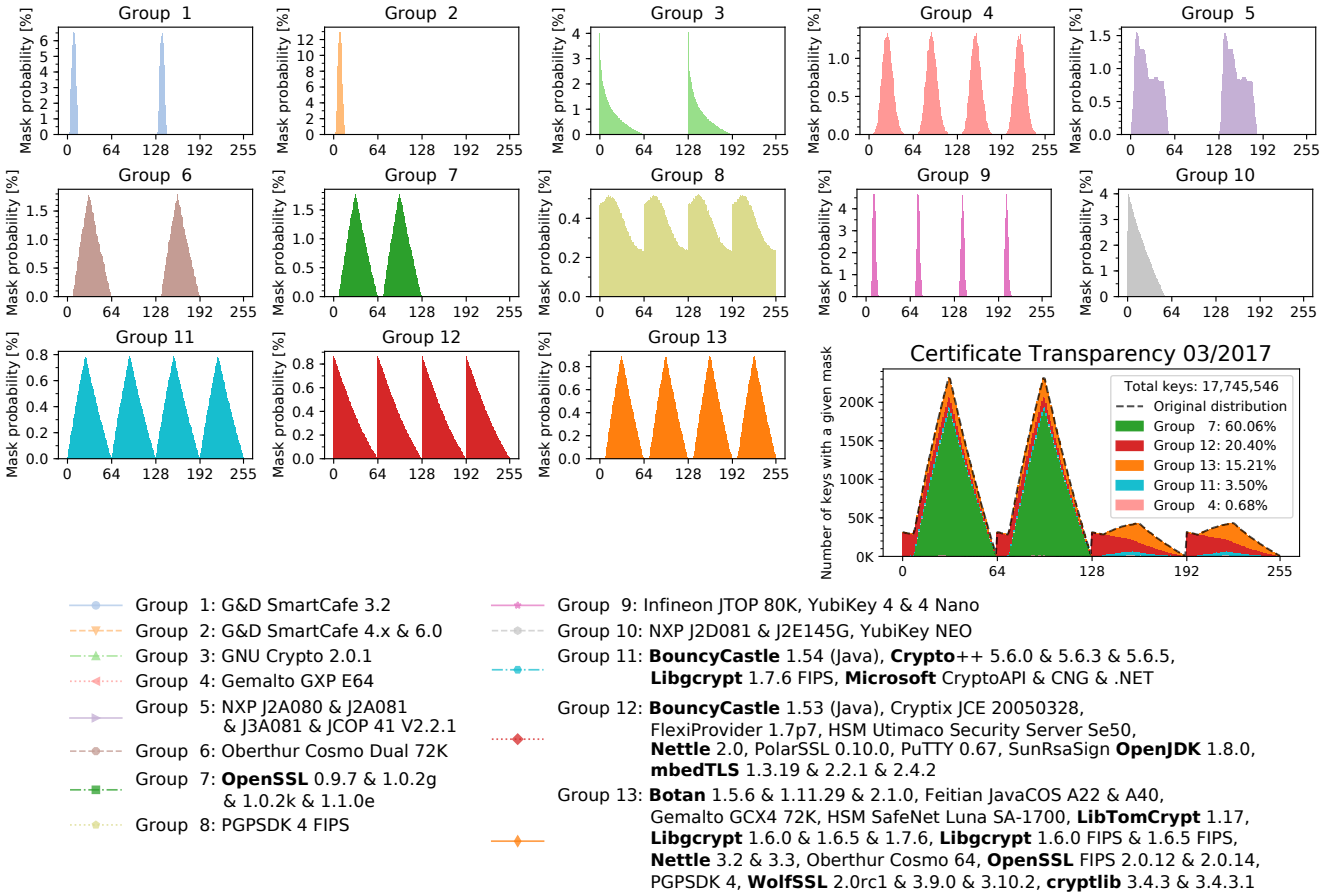

Figure 3: The reference distributions of the key mask from each library are used to compute the probability with which the given library contributes to the overall distribution of keys in the measured sample. The distribution of keys submitted to Certificate Transparency logs during March 2017 likely contains keys from a mix of distributions as given in the last picture. When we scaled each distribution accordingly and plotted the bars on top of each other (note – the bars are not overlapping), the fit is visually close (the original distribution is given by a black dashed outline and matches the tops of the approximated bars).

keys and the reference distributions are empirical, a precise solution may not exist, due to the presence of noise (see Section 3.4.1).

We chose the linear least squares method constrained to non-negative coefficients (non-negative least squares fit or NNLSF) implemented in Java [28] based on the algorithm by Lawson and Hanson [30] to find an approximate solution to the overdetermined system of equations. The solution is the estimated prior probability $\hat{P}(G_j)$ for each group G_j. The method numerically finds a solution that minimizes the sum of squared errors $(P(K_i) - \hat{P}(K_i))^2$ over all n mask values K_i, where $P(K_i)$ is the idealized probability of mask K_i (obtained from the dataset) and $\hat{P}(K_i)$ is the estimated probability, given by substituting the real group probability $P(G_j)$ in Equation 2 with the estimated group probability $\hat{P}(G_j)$.

3.3 Key classification

We classify the keys according to their origin using the Bayes' rule (Section 2.3). When compared to the approach of [36], we use the estimated prior probabilities for a more precise classification. In a classification of a single key, the groups are ordered by the value of the conditional probability $P(G|K)$. The group G with the highest probability is the most likely origin of the key.

3.4 Evaluation of accuracy

We are interested both in the accuracy of the prior probability estimation (given as the expected error in the estimation, Table 1) and in the average correctness of the overall classification process (given as the proportion of keys that were correctly classified, Table 2). For the measurement, we repeatedly simulate large datasets according to different distributions, add noise and perform our method.

3.4.1 Random noise. Even if keys in a large dataset were generated with the same library across many users, the overall distribution will not match our reference distribution exactly, due to the non-deterministic nature of key generation. This contributes to a

random noise in the data. We achieve such a noise in our simulations by generating masks non-deterministically (i.e., instead of using reference distributions in place of data, we randomly sample masks according to the distribution).

3.4.2 Systematic noise. Our analysis does not cover all existing libraries used on the Internet. However, it is quite likely that algorithms used by unknown libraries are similar to those already known (e.g., consider the size of Group 13). In such a case, the library would belong to one of our groups and the only error of the estimation would be in the interpretation of the results – the library is not correctly labeled as a part of our group. Yet still, there may exist groups with profiles that do not match any of our known groups, hence keys generated from these implementations would add systematic noise to the profile of the sample. In our simulations, we create a group representing all unknown distributions. The group profile is chosen randomly in each experiment. To simulate the presence of keys from this group, we modify the prior probability of the simulation to include a certain percentage (e.g., ranging from 0% to 3% in Tables 1 and 2) of keys to be sampled from the distribution. For example, 3% of systematic noise represents the situation where 3% of the keys in the sample originate from an unknown distribution, not covered by our analysis and belonging to a completely different, never seen before, group.

3.4.3 Simulation scenarios. We considered several distributions of prior probability library usage:

Evenly distributed probabilities match the approach in [36], however, we face an additional task of first estimating the probabilities from the simulated data. Furthermore, our mask does not use one of the original features (Section 2.1).

We also assign *random prior probabilities* to the groups in a different scenario – each group is assigned a uniformly chosen real number from 0 to 1 and the numbers are normalized to sum to 1.

Real-world popularities of libraries are better characterized by a *geometric distribution* – one source dominates (e.g., 50% in our case) and other are exponentially less probable. We additionally ensure that each group has a probability at least 2%. This way, even very rare sources are not completely excluded from the analysis, even if the library is outdated (e.g., PGP SDK 4) or the hardware is very old (e.g., Gemalto GXP E64 smartcard from 2004). We also test the geometric distribution for different permutations of the groups – while in TLS, OpenSSL is always the most probable, in our tests each group may take the first place.

Finally, we simulate the data according to the prior *probabilities extracted from TLS datasets*. We add deviations to the probabilities to simulate subtle changes in the popularity of libraries.

3.4.4 Accuracy of prior probability estimation. The accuracy of the prior probability estimation is given as the expected error in the resulting estimation. The summary is given in Table 1.

We consider the average error (the expected error in percentage points (*pp*) for each group probability in each experiment) and the average worst error (the expected error in *pp* for the worst result in a given experiment). As an example, if the real probabilities are 60%, 30%, and 10%, and we estimate them as 61%, 32%, and 7%, the average error of the experiment is $(1+2+3)/3 = \pm 2$ *pp* and the worst error is ± 3 *pp*. The averages in Table 1 are given for 100 experiments, each simulating one million keys. We considered distinct scenarios (Section 3.4.3) and levels of systematic noise (Section 3.4.2).

	Estimation error (in percentage points)							
Noise:	0%	1%	2%	3%	0%	1%	2%	3%
Distribution	Average error				Average worst error			
Even	0.19	0.37	0.63	0.90	0.73	1.71	3.33	5.07
Random	0.19	0.37	0.61	0.84	0.78	1.74	3.25	4.68
Geometric	0.18	0.38	0.63	0.91	0.71	1.70	3.33	4.97
TLS	0.17	0.39	0.66	0.94	0.65	1.78	3.49	5.16

Table 1: Accuracy of prior probability estimation for different types of distributions and different amount of systematic noise. The average error gives the expected error of prior probability estimation for each group in percentage points (pp). The average worst error gives the expected value of the largest error in each experiment. E.g., when the keys were generated from a TLS-like distribution with 1% of systematic noise added, the probability of each group differed by ±0.39 pp on average and the worst estimation was off by ±1.78 pp on average.

	Classification accuracy (in %) with noise							
Noise:	0%		1%		2%		3%	
Guess:	1st	2nd	1st	2nd	1st	2nd	1st	2nd
Even	33.4	53.2	33.3	52.9	32.9	52.4	32.3	51.8
Random	45.5	67.5	46.1	67.8	45.0	66.7	43.9	65.1
Geometric	81.1	95.2	82.5	95.0	80.3	94.6	80.9	94.3
TLS	94.8	98.7	94.6	98.6	94.4	98.4	94.3	98.3

Table 2: Accuracy of key origin classification when prior probability estimates are included in the method for different types of distributions and different amount of systematic noise. The values are in percents. E.g., when the keys were generated from a TLS-like distribution with 1% of systematic noise added, for 94.6% of the keys, the original library was correctly identified on the first guess and 98.6% of keys were correctly labeled by the first or the second most probable group.

3.4.5 Accuracy of the overall classification process. The accuracy of key classification is given as the proportion of keys that were correctly classified as the first guess or at least the second guess. The values in Table 2 are given in percents.

Tables 1 and 2 refer to the same set of simulations. The prior probability estimation is performed first. The results show that the classification is quite robust even in the case of errors in prior probability estimations at a level of 5 percentage points, since the success of the classification is not affected dramatically.

When compared to the approach of [36], the average accuracy increased for other than the even distribution of groups. However, the classification accuracy is improved mostly for the more probable groups and the less probable libraries may be classified incorrectly more frequently than before.

3.5 Additional accuracy considerations

Some reference distributions can be approximated by a combination of other reference distributions, similarly as the distribution observed in a dataset can be obtained as a combination of reference distributions. An example of this phenomenon at its worst is the close match of Group 11 (Microsoft libraries) as a combination of 41.3% of Group 13, 30.6% of Group 8, 22.7% of Group 4 and a small portion of other groups. The situation for all groups is illustrated in Figure 4, with the most notable groups enlarged. Group 13 has the next closest match, however the error is much larger. Group 7 (OpenSSL) cannot be obtained as a combination of other groups.

As a result, the prior probability estimation process may interchange the distribution of Group 11 for a mixture of other distributions or vice versa. Currently, we do not detect such events automatically, since an additional user input would be needed.

When considering the results, the domain must be taken into account. E.g., according to our measurement, around 1% of keys in some samples of TLS keys originate from Group 8 (PGP SDK 4 FIPS). Since the presence of the library in TLS is highly unlikely and no other known implementation has the same (quite uncommon) algorithm, we must conclude that this is an error in the estimation. We suspect the error is due to the aforementioned approximation of Group 11. However, there may exist different approximations of the group, hence we cannot simply substitute the suspected ratio.

We hypothesize that such errors could be avoided if the prior probability estimation would start from a very rough approximation of the probabilities supplied by the user (we use evenly distributed groups) as the starting guess of the NNLSF method. A more resolute solution would remove groups from the analysis if they are unlikely to occur in an examined domain according to empirical evidence.

4 RESULTS ON RELEVANT DATASETS

Rough estimates of popularity for some cryptographic libraries were provided in [36] for TLS, CT and PGP, but with relatively high expected errors. The improvement of accuracy in our work allows for a better inspection of datasets, including the detection of transient events. We also processed significantly more datasets, including the archived ones.

4.1 Data preparation

We used a wide range of datasets for our analysis. Due to different formats, we pre-process all data into a unified intermediate format. For all datasets, only keys with unique moduli were considered.

4.1.1 Censys TLS scan. Censys [15] performs a full IPv4 address space scan of TCP port 443 on a weekly basis [4]. The dataset contains historical scans back to 2015-08-10 when the first scan was performed and continues to present. Each scan is a full snapshot, independent from all other scans, containing all raw and post-processed data from the scan in the form of JSON and CSV files, compressed by LZ4 algorithm [5]. Some snapshots are only a few days apart and some larger gaps occur, but overall the weekly periodicity is prevalent.

Censys scanner tries to perform a TLS handshake with the host being scanned, respecting the IP blacklist maintained by Censys. The latest scan tried to contact 53M hosts.

4.1.2 Censys Alexa 1M. The dataset [3] has the same properties as the Censys IPv4 dataset (with respect to periodicity and format). It contains processed TLS handshakes with the top 1 million websites according to the Alexa ranking. The dataset also provides an insight into a specific portion of the Internet certificates, which are otherwise hidden from ordinary IPv4 scans because of the use of Server Name Indication (SNI) TLS extension. SNI enables the web server to multiplex X.509 certificates on a single IP address, because the client sends the desired host name directly in the TLS handshake. Simple TLS handshake returns only one, default virtual host certificate, hence other virtual hosts are hidden from the scan. Moreover, the default certificate is usually generated during server installation (if not overridden later) and thus does not have to be relevant to the context.

4.1.3 Rapid7 Sonar SSL. Project Sonar [6] performs a regular scan of IPv4 SSL services on TCP port 443. The dataset includes both raw X.509 certificates and processed subsets. It contains snapshots taken within a time frame of maximum 8 hours. It ranges from 2013-10-30 to the present (still active) with many samples. The files with certificates are incremental, so the scan from a particular day contains only new certificates – not yet seen in the preceding scan. We transform the increments into snapshots. The scanning periodicity varies, making the analysis more complicated. The project also maintains an independent IP address blacklist that evolves in time. Additionally, the scanner code evolves (cipher suite selection, bug fixes, methodology fixes) causing fluctuations in the data.

4.1.4 HTTPS Certificate Ecosystem. IPv4 TLS scanning dataset [18] ranging from 2012-06-10 to 2014-01-29. It is essentially the same as the Sonar SSL dataset with respect to the format and the properties. This dataset contains one host-to-certificate fingerprint mapping file for each scan and one big certificate database for the whole dataset. The periodicity varies a lot. There are many snapshots only two days apart, as well as large gaps between samples, up to 50 days. We recoded the dataset to the Sonar SSL format, with an incremental certificate database. We then transformed it to the full snapshot format identical as for Sonar SSL.

4.1.5 Certificate Transparency. The specification of CT (RFC 6962) allows retrieving an arbitrary range of entries from a log. We processed all entries in logs maintained by Google up to May 2017. All entries must be submitted with all the intermediate certificates necessary to verify the certificate chain up to a root certificate published by the log. We process only the leaf certificates. Since the logs are append-only, there is no reliable way of knowing whether an older certificate is still active (the validity period gives an upper estimate), hence we do not have a sample of all certificates in use for a given date. Instead, we process incremental samples – all certificates submitted during a specific period (a day or a week).

4.1.6 Client SSH keys – GitHub. GitHub gives users SSH-authenticated access to their Git repositories. Developers upload their public SSH keys. One user can have no, one or more SSH keys. GitHub provides an API to list all the registered users and another endpoint allows downloading SSH keys on a per-user basis. We downloaded a list of almost 25M GitHub users with almost 4.8M SSH keys found. The scan was performed in February 2017 and took 3 weeks to finish on a commodity hardware. We implemented a

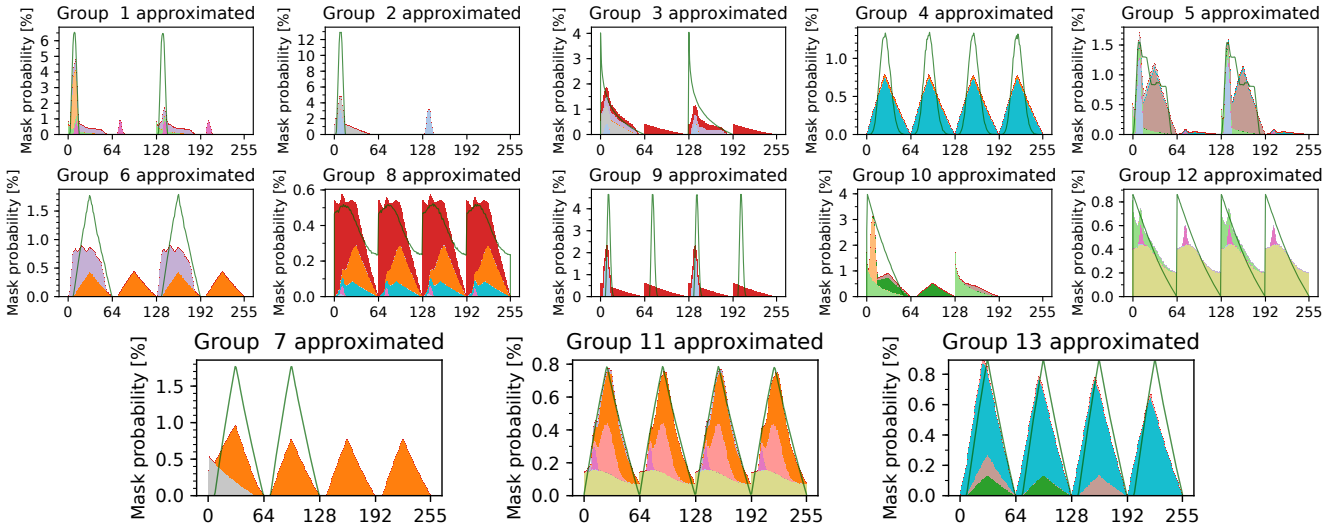

Figure 4: Some distributions may be interchanged for a mix of other distributions. We used our method to approximate each of the 13 distributions using only the remaining 12 distributions. The graphs show the original distribution as a green outline and the combination of libraries that minimizes the sum of squared distances is visualized by stacking the scaled distributions on top of each other. The results show that Group 7 (OpenSSL) cannot be easily approximated by other groups (the process leads to a large squared differences in the distributions that will not be permitted by the NNLSF method). However, Group 11 (with Microsoft libraries) can be simulated relatively closely by a combination of other distributions. Hence, when a real world distribution contains keys from Group 11, the method may misattribute the keys as coming from a specific mixture of libraries instead.

custom multi-threaded crawler for this purpose, downloading user list, SSH keys, parsing them and producing a file for classification.

4.1.7 Pretty Good Privacy (PGP). PGP key servers play an important role in the PGP infrastructure as public registers of public PGP keys. The PGP servers synchronize among themselves periodically. We downloaded a dump of the database in April 2017, parsed it and extracted RSA master and sub-keys for the analysis. Anyone can upload a valid PGP public key to the key server and download the key later. This has to be taken into account during analysis. Anyone can generate thousands of keys and upload them to the key server, which would skew a statistics. This actually happened when a group called Evil 32 [29] generated a new PGP key for thousands of identities in the PGP server with a collision on the short key ID to demonstrate the weakness of using a short 32-bit identifier in the PGP ecosystem.

4.2 Internet-wide TLS scans

Various projects performed Internet-wide scanning since 2010, with different periods, frequencies and scanning techniques. We extracted unique RSA keys from certificates collected by EFF SSL Observatory (only two scans), HTTPS Ecosystem (07/2012-02/2014), Rapid7 SonarSSL (11/2013-05/2017) and Censys IPv4 TLS (08/2015-05/2017) scans. The processing is described in Section 4.1.

The overlapping portions of the different scans provide a good match except for Group 11 (Microsoft libraries) in the Rapid7 Sonar SSL scan between 11/2013 to 06/2015. The significant decrease of Microsoft libraries is caused by an improper implementation of the TLS v1.2 handshake by the scanning software, resulting in exclusion of a significant portion of Microsoft IIS servers for 18 months as confirmed by Project Sonar authors.

Figure 5 shows the absolute number of unique RSA keys attributed by us to every classification group of cryptographic libraries. OpenSSL (Group 7) is increasingly more popular, also relatively to other libraries. As of May 2017, there are about 8 million active unique RSA keys generated by OpenSSL. Group 11 that contains Microsoft libraries is relatively stable since 2012 starting with 2M, rising to 2.4M in 2014 and then slightly decreasing to 2.2M keys in 2016. Since there are several changes in the data collection methodology and software, it is difficult to make a conclusion about the significance of the numbers. However, the data indicates a comparably stable number of keys originating from the group.

The large Group 13 (containing Nettle, OpenSSL FIPS, and WolfSSL among others) used to be the third most common library with 0.4-0.5M keys, but was gradually matched by Group 12 (containing OpenJDK and mbedTLS) in year 2016. Both groups now have an almost equal share of about 0.5 M unique keys.

The last somewhat significant group is Group 8 with about 1% of keys, slightly decreasing in popularity since 2016. The group contains only the PGPSDK 4 FIPS implementation, which is unlikely to be so popular in TLS. There either exists a different popular library with a similar prime generation algorithm (not included in the set of libraries studied by us), or a portion of the dataset was misattributed to the library due to a similarity of a combination of profiles, as explained in Section 3.5.

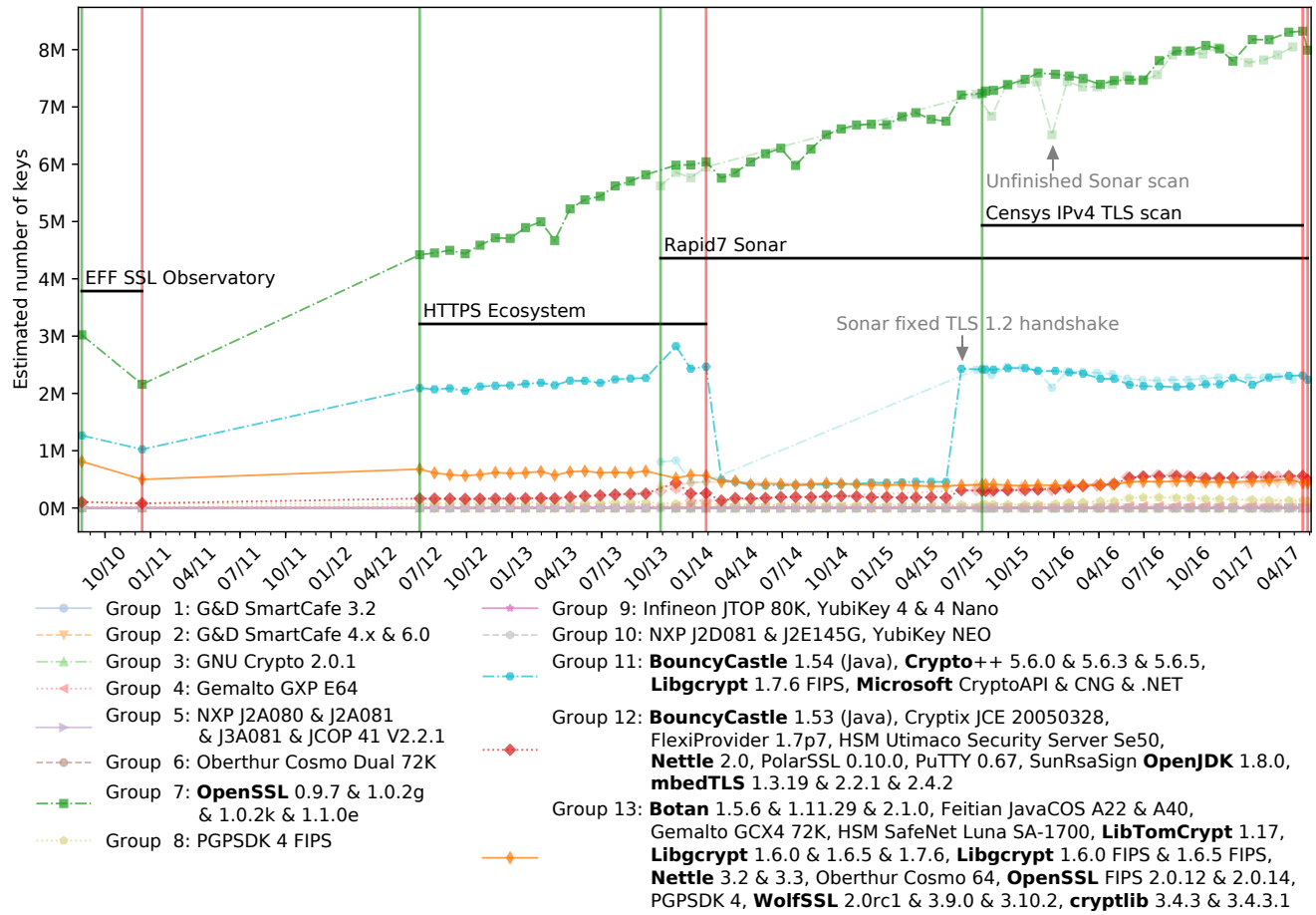

Figure 5: The combined results from scans of TLS services in the whole IPv4 space as provided by four independent datasets, given with one-month granularity. An absolute number of unique keys as attributed to different groups by our method are shown. The sudden "jump" for Group 11 (Microsoft libraries) in SonarSSL in 06/2015 is caused by an improper implementation of TLS 1.2 handshake in the scanning software, resulting in an exclusion of a significant portion of Microsoft IIS servers for 18 months.

4.3 Popularity between usage domains

Although the TLS ecosystem is the most frequently studied one, large datasets of RSA keys exist for other usage domains. We analyzed and compared the relative popularity of cryptographic libraries as of March 2017 for Internet-wide TLS scans (Censys), obtained from the 1 million most popular domains according to the Alexa survey, and the certificates uploaded to all Google's Certificate Transparency logs during that month. We also present the TLS keys as of December 2010 to illustrate the progress in time. Additionally, SSH authentication keys of all GitHub users and all keys from PGP key servers were analyzed. The differences are shown in Figure 6.

The analysis shows significant differences among the usage domains. The GitHub SSH dataset is clearly dominated by OpenSSL with more than 96% – the default library behind *ssh-keygen* utility from OpenSSH software. Fewer than 3% belong to Group 12, which contains the popular SSH client PuTTY for Microsoft Windows.

The PGP keys are generated mostly by Group 13 (containing *Libgcrypt* from the widely used GnuPG software) with about 85% share, followed by OpenSSL with approximately 11%.

4.4 TLS to CT comparison

According to a survey based on IPv4 scans, Certificate Transparency (CT) and a large set of active domain names [37], the combination of CT and IPv4 scans provides a representative sample of the Internet. We are interested in the differences between the methodologies.

An interesting popularity distribution can be observed from CT logs. CT has been used on a large scale since 2015, with the first logs launching in 2013. The logs now contain almost an order of magnitude more certificates than those reachable by direct IPv4 TLS scans. Due to the validation of TLS certificates performed by all modern browsers, all valid certificates used for TLS are now present in CT, but also more. CT logs also contain TLS certificates hidden from IPv4-based scans due to Server Name Indication (SNI)

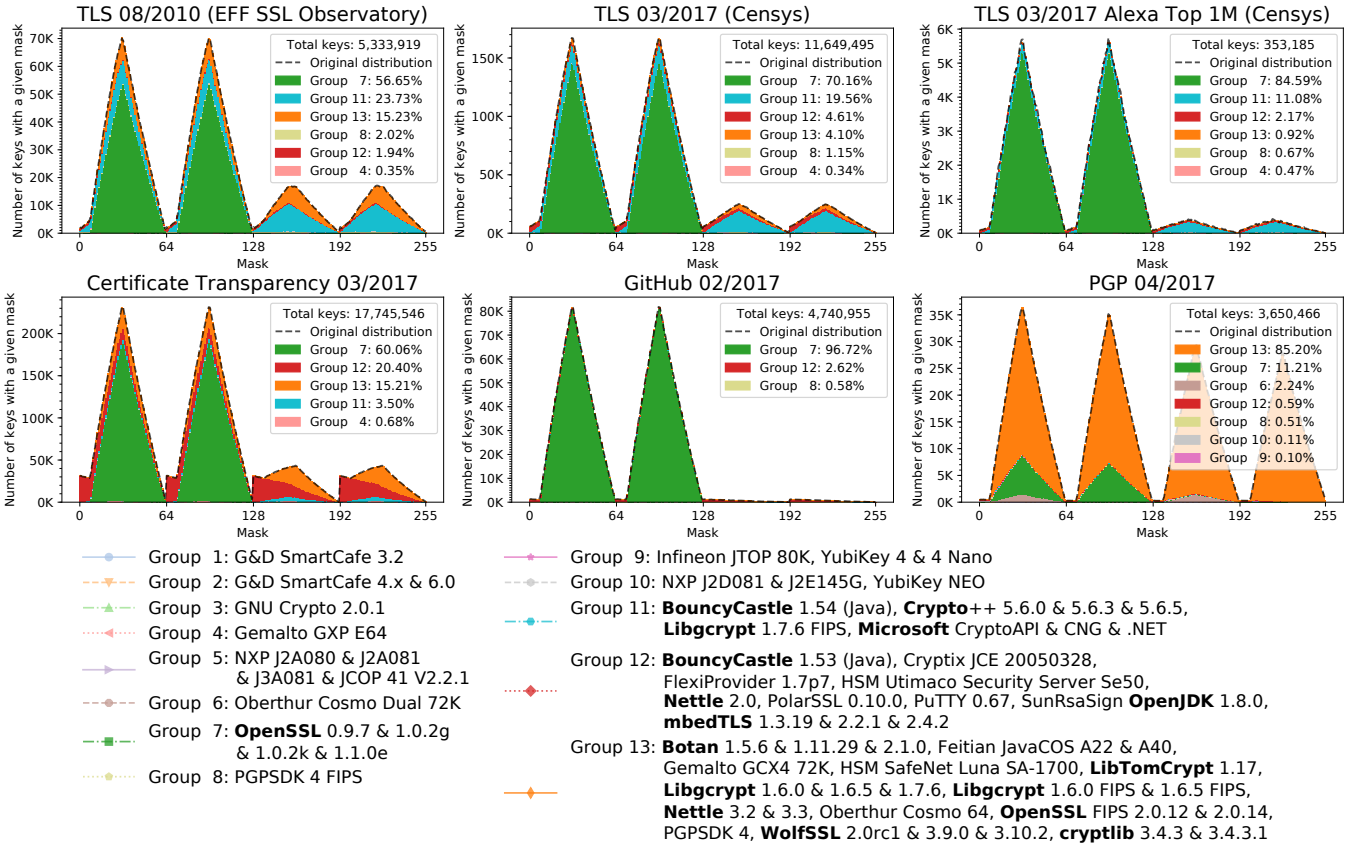

Figure 6: Library share in different usage domains. The sources responsible for at least 0.1% of all keys in a particular scan of the domain are listed in the legend. OpenSSL library dominates in all domains, except for the PGP dataset.

TLS extension. Additionally, certificates never seen in TLS or not intended for TLS can be submitted to the logs. According to a study of the CT landscape [22], almost 95% of certificates stored in CA operated logs are also seen in CT logs operated by Google (Pilot, Icarus, Rocketeer, Skydiver, Aviator) – we therefore use these logs with newly inserted certificates during certain time frames (a day, a week, a month) to perform our analysis.

We compare selected results for certificates submitted to CT during March 2017 with a Censys scan from the same month in Figure 6. While OpenSSL is again the most common library in CT, it is responsible only for about 60% of unique RSA keys, where the Censys scan contains about 70% of the same. Microsoft libraries (Group 11) are in a minority with 3.5% in CT, whereas they are responsible for almost 20% in TLS. The longer validity of certificates generated by Microsoft software (especially when compared to certificates produced by Let's Encrypt CA with 3-month validity) is a potential reason, with SNI multiplexing being another one. Groups 12 and 13 are relatively common in CT with 20% and 15%, respectively, whereas both are below 5% in TLS.

4.5 Detection of transient events

We used our method to estimate the proportion of libraries for keys newly submitted every week to Google's CT servers between

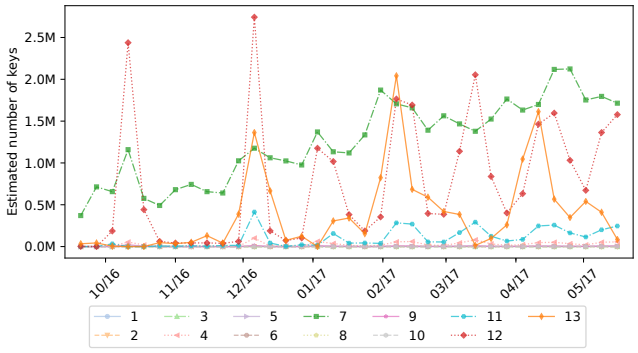

Figure 7: The number of keys from distinct groups added to CT weekly, found in certificates issued by Let's Encrypt CA.

October 2016 and May 2017, limited to certificates issued by Let's Encrypt CA, as shown in Figure 7. The number of certificates added every week fluctuates significantly, as well as the responsible libraries. Only a relatively small number of keys from Group 11 were inserted when compared to the number of certificates in active use found by TLS scans. This suggest that Microsoft libraries are less

likely to be used with Let's Encrypt software. Interestingly, there is a certain periodicity between such certificates being submitted.

Some periodic monthly insertion events are also visible for Group 12 (OpenJDK, Bouncy Castle before v. 1.54, mbedTLS, etc.) and bi-monthly for Group 13 (OpenSSL FIPS, WolfSSL, etc.). Most Let's Encrypt certificates from the events are reissued after 60 days.

5 RELATED WORK

Only very few prior publications are concerned with the identification of the library responsible for generating an RSA key. Except for [36] (the work we directly improve on), the task was done by [32], who observed that particular biases in private keys generated by OpenSSL can be also seen in the majority of keys that were found in TLS scans and factored by [1, 23, 24]. However, the method only worked because of the knowledge of the private primes. Furthermore, the keys were generated with insufficient entropy due to bad random generators. Hence the technique can be extended neither to all keys generated by OpenSSL, nor to other libraries.

The popularity of a library can be also estimated from the positive ratings (stars or likes) of open-source repositories, such as those hosted on GitHub. However, this seems to be a very poor method – OpenSSL only has four times as many stars as mbedTLS and closed-source libraries like Microsoft CAPI/CNG cannot be compared this way at all.

Server fingerprints were used to probabilistically determine the operating system, or even the versions of the deployed software [34, 35]. Indeed, the estimates on the number of servers running Microsoft OS published by [33] matches the results of our analysis of a scan of the Alexa Top 1 million domains. A similar analysis was performed for software packages handling the SSH connection [8] mostly served by Dropbear and OpenSSH, confirming the dominance of OpenSSH-based software.

Debian-based Linux distributions offer public statistics about the popularity of software packages as a part of a quality assurance effort [7]. The results are based on a relatively high number of users (almost 200K) and provide an insight into the number of package installation, yet they cannot capture the number of keys in use. The libraries used to validate SSL certificates in non-browser client software were surveyed in [21].

A direct identification of software packages running on other cores in a cloud environment based on cache side-channels was demonstrated by [26, 27]. The measurement requires a local presence, does not scale and cannot be used on archived datasets. However, it recovers not only the library, but also a particular version.

Measurements and analyses of the TLS ecosystem have a long history with large scale scans starting in 2010 with the EFF SSL Observatory project [2], followed by analyses of both valid certificates [13, 16, 17, 19, 20] (the majority of papers) as well as invalid ones [12]. The significant increase of popularity of Certificate Transparency servers now provides a view of the certificates that are otherwise unreachable via IP address based scanning [37]. Researchers usually focus on the properties of the certificates (e.g., validity period) or the certificate chain extracted from the TLS handshakes. Chosen cryptographic algorithms and key lengths were also analyzed [15, 25], showing that more than 85% of currently valid certificates use the RSA algorithm – making our method based on RSA keys representative of the ecosystem.

The client SSH authentication keys extracted from GitHub were previously collected and analyzed [9, 14] with a focus on the algorithms, key lengths, and presence of weak keys, detecting keys generated from OpenSSL with insufficient entropy.

6 CONCLUSIONS

A wide-scale accurate measurement of the popularity of cryptographic libraries is an important precursor for a security analysis of the Internet ecosystem, such as an evaluation of resilience against security bugs. Yet so far, it was based only on proxy measurements, like the popularity of web server implementations. We proposed a measurement method based on statistical inference, which finds a match between the observed distribution of keys on the Internet and a specific proportion of reference distributions of RSA public keys extracted from cryptographic libraries. Our method does not rely on active communication with a server implementation, hence it also works when proxy information is not available, such as for SSH client keys, where direct scanning of clients is not performed. The analysis is possible thanks to the recently discovered biases in the implementations of RSA public key generation [36].

The results show an overall increasing reliance on OpenSSL – its share grew from 56% to 70% between the years 2010 and 2017 as observed from keys used by TLS servers. The prevalence of OpenSSL reaches almost 85% within the current Alexa top 1M domains and more than 96% for client-side SSH keys as used by GitHub users. The usage trends of Microsoft libraries are mostly stable with a share of around 20% for TLS serves and a 10% share of the Alexa top 1M domains. The GnuPG Libgcrypt library and statistically similar implementations are responsible for 85% of all PGP keys. Certificate Transparency logs provide a different ratio of libraries for recently added certificates than Internet-wide scans – OpenSSL is down to 60%, Microsoft is at only 3.5% (probably due to longer validity of certificates) and the remaining libraries account for more than 35% (while their share in IPv4 TLS scans is lower than 10%).

This method can also capture short-term events, when incremental datasets are examined (e.g., daily changes). We observed that many certificates from specific libraries were submitted to Certificate Transparency logs periodically, coinciding with the validity of Let's Encrypt certificates. Our measurement also revealed an inconsistency between historical datasets, caused by a bug in the scanning software of Project Sonar, which led to an omission of more than a million Microsoft servers from IPv4 TLS scans during the period of 18 months.

ACKNOWLEDGMENTS

We would like to thank our colleagues for fruitful discussions, especially Marek Sys and Stanislav Katina. We acknowledge the support of the Czech Science Foundation under project GA16-08565S. The access to the computing and storage resources of National Grid Infrastructure MetaCentrum (LM2010005) is also greatly appreciated. Vashek Matyas thanks Red Hat Czech and CyLab, Carnegie Mellon University for a supportive sabbatical environment during some of his work on this paper.

REFERENCES

[1] DSA-1571-1 openssl – predictable random number generator, 2008. [cit. 2017-09-20]. Available from https://www.debian.org/security/2008/dsa-1571.

[2] The EFF SSL Observatory, 2010. [cit. 2017-09-20]. Available from https://www.eff.org/observatory.

[3] Censys TLS Alexa Top 1 Million Scan, 2015. [cit. 2017-09-20]. Available from https://censys.io/data/443-https-tls-alexa_top1mil.

[4] Censys TLS Full IPv4 443 Scan, 2015. [cit. 2017-09-20]. Available from https://censys.io/data/443-https-tls-full_ipv4/historical.

[5] LZ4 Extremely Fast Compression algorithm, 2015. [cit. 2017-09-20]. Available from http://www.lz4.org/.

[6] Rapid 7 Sonar SSL full IPv4 scan, 2015. [cit. 2017-09-20]. Available from https://scans.io/study/sonar.ssl.

[7] Debian quality assurance: Popularity contest statistics, 2017. [cit. 2017-09-20]. Available from https://qa.debian.org/popcon.php.

[8] Albrecht, M. R., Degabriele, J. P., Hansen, T. B., and Paterson, K. G. A surfeit of SSH cipher suites. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), CCS '16, ACM, pp. 1480–1491.

[9] Barbulescu, M., Stratulat, A., Traista-Popescu, V., and Simion, E. RSA weak public keys available on the Internet. In *International Conference for Information Technology and Communications* (2016), Springer-Verlag, pp. 92–102.

[10] Bernstein, D. J. Cache-timing attacks on AES, 2005. [cit. 2017-09-20]. Preprint available at https://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[11] Brumley, D., and Boneh, D. Remote timing attacks are practical. In *Computer Networks* (2005), vol. 48, Elsevier, pp. 701–716.

[12] Chung, T., Liu, Y., Choffnes, D., Levin, D., Maggs, B. M., Mislove, A., and Wilson, C. Measuring and applying invalid SSL certificates: The silent majority. In *Proceedings of the 2016 ACM on Internet Measurement Conference* (2016), ACM, pp. 527–541.

[13] Clark, J., and van Oorschot, P. C. SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE Symposium on Security and Privacy* (2013), IEEE, pp. 511–525.

[14] Batch-GCDing Github SSH Keys, 2015. [cit. 2017-09-20]. Available from https://cryptosense.com/batch-gcding-github-ssh-keys/.

[15] Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., and Halderman, J. A. A search engine backed by internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 542–553.

[16] Durumeric, Z., Bailey, M., and Halderman, J. A. An internet-wide view of internet-wide scanning. In *Proceeding of USENIX Security Symposium* (2014), pp. 65–78.

[17] Durumeric, Z., Kasten, J., Adrian, D., Halderman, J. A., Bailey, M., Li, F., Weaver, N., Amann, J., Beekman, J., Payer, M., et al. The matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (2014), ACM, pp. 475–488.

[18] Durumeric, Z., Kasten, J., Bailey, M., and Halderman, J. A. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 13th Internet Measurement Conference* (2013).

[19] Durumeric, Z., Kasten, J., Bailey, M., and Halderman, J. A. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 2013 ACM Internet Measurement Conference* (2013), ACM, pp. 291–304.

[20] Felt, A. P., Barnes, R., King, A., Palmer, C., Bentzel, C., and Tabriz, P. Measuring HTTPS adoption on the web. In *26th USENIX Security Symposium* (2017), USENIX Association, pp. 1323–1338.

[21] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., and Shmatikov, V. The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (2012), ACM, pp. 38–49.

[22] Gustafsson, J., Overier, G., Arlitt, M., and Carlsson, N. A first look at the CT landscape: Certificate Transparency logs in practice. In *Proceedings of the 18th Passive and Active Measurement Conference* (2017), Springer-Verlag, pp. 87–99.

[23] Hastings, M., Fried, J., and Heninger, N. Weak keys remain widespread in network devices. In *Proceedings of the 2016 ACM on Internet Measurement Conference* (2016), ACM, pp. 49–63.

[24] Heninger, N., Durumeric, Z., Wustrow, E., and Halderman, J. A. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Proceeding of USENIX Security Symposium* (2012), vol. 8.

[25] The ICSI Certificate Notary, 2017. [cit. 2017-09-20]. Available from https://notary.icsi.berkeley.edu/.

[26] Inci, M. S., Gulmezoglu, B., Eisenbarth, T., and Sunar, B. Co-location detection on the cloud. In *International Workshop on Constructive Side-Channel Analysis and Secure Design* (2016), Springer-Verlag, pp. 19–34.

[27] Irazoqui, G., Inci, M. S., Eisenbarth, T., and Sunar, B. Know thy neighbor: crypto library detection in cloud. *Proceedings on Privacy Enhancing Technologies 2015*, 1 (2015), 25–40.

[28] Kaminsky, A. Parallel Java 2 library (PJ2), 2017. [cit. 2017-09-20]. Available from https://www.cs.rit.edu/~ark/pj2.shtml.

[29] Klafter, R., and Swanson, E. Evil 32, 2015. [cit. 2017-09-20]. Available from https://evil32.com.

[30] Lawson, C. L., and Hanson, R. J. *Solving Least Squares Problems*. SIAM, 1995.

[31] Lazar, D., Chen, H., Wang, X., and Zeldovich, N. Why does cryptographic software fail?: a case study and open problems. In *Proceedings of 5th Asia-Pacific Workshop on Systems* (2014), ACM, pp. 1–7.

[32] Mironov, I. Factoring RSA Moduli II. [cit. 2017-09-20]. Available from https://windowsontheory.org/2012/05/17/factoring-rsa-moduli-part-ii/.

[33] NetCraft April 2017 Web Server Survey, 2017. [cit. 2017-09-20]. Available from https://news.netcraft.com/archives/2017/04/21/april-2017-web-server-survey.html.

[34] NetCraft operating system detection, 2017. [cit. 2017-09-20]. Available from http://uptime.netcraft.com/accuracy.html#os.

[35] Nmap Remote OS Detection, 2017. [cit. 2017-09-20]. Available from https://nmap.org/book/osdetect.html.

[36] Svenda, P., Nemec, M., Sekan, P., Kvasnovsky, R., Formanek, D., Komarek, D., and Matyas, V. The million-key question — Investigating the origins of RSA public keys. In *Proceeding of USENIX Security Symposium* (2016), pp. 893–910.

[37] VanderSloot, B., Amann, J., Bernhard, M., Durumeric, Z., Bailey, M., and Halderman, J. A. Towards a complete view of the certificate ecosystem. In *Proceedings of the 2016 ACM on Internet Measurement Conference* (2016), ACM, pp. 543–549.

A ADDITIONAL RESULTS

Table 3 shows the sources considered in the analysis, together with the relevant biases.

Figure 8 shows the number of keys attributed by us to different cryptographic libraries in certificates from the Alexa Top 1 million domains collected by Censys. The number of OpenSSL keys is rising and the percentage of keys coming from Microsoft implementations is much smaller than in general TLS scans.

Previous analyses of Internet-wide TLS scans [13, 16, 17, 19] compared various properties of certificates. Valid and invalid certificates were compared by [12], showing that the majority of certificates found by scans are invalid and have interesting properties.

We compared self-signed certificates to certificates signed by third parties in historical datasets from HTTPS Ecosystem and Rapid7 Project Sonar. Figure 9 shows a significant difference in the keys coming from such certificates. Most notably, Microsoft keys are found in self-signed certificates less commonly than OpenSSL keys. As explained in Section 4.2, the decrease in the number of certificates between 11/2013 to 06/2015 is caused by an improper implementation of the TLS v1.2 handshake used by Project Sonar.

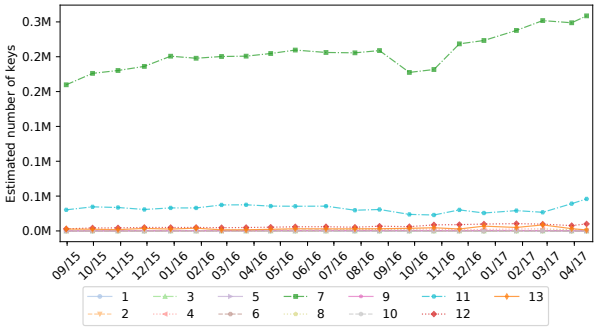

Figure 8: More domains from the Alexa Top 1M list use OpenSSL (Group 7) now than in 2015. Note that the number of keys does not sum to 1M already in the original dataset collected by Censys. Some websites do not support HTTPS [20] or the specific cipher-suite used by the Censys scanner.

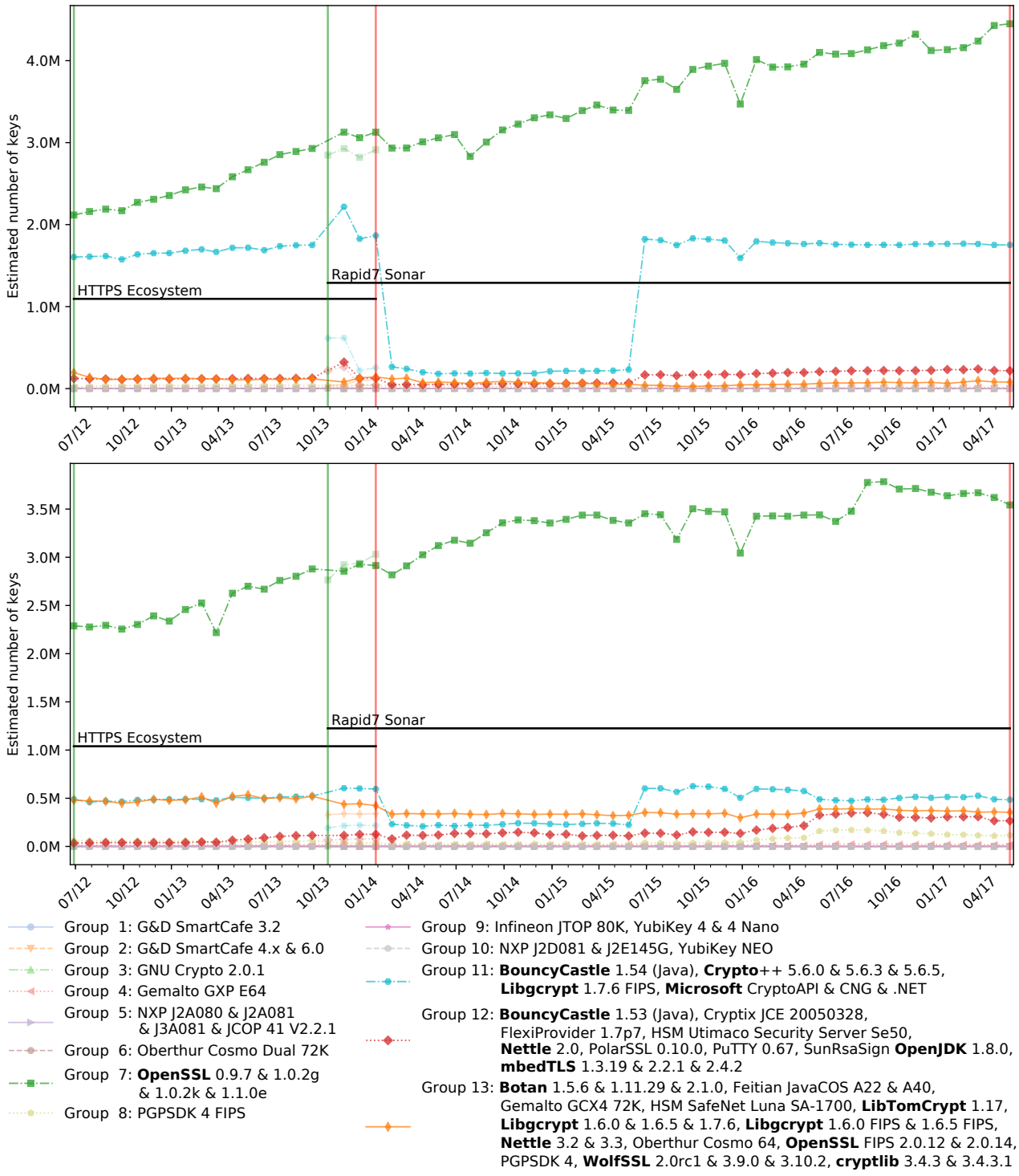

Figure 9: Comparison of library popularity for keys coming from certificates signed by a third party (top) and self-signed certificates (bottom). Self-signed certificates are dominated by OpenSSL. More than 50% of OpenSSL keys observed in 2012 were found in self-signed certificates. For OpenSSL, the number of not self-signed certificates rose faster than the number of self-signed certificates, and significantly more OpenSSL certificates are now signed by a third party. Fewer than 25% Microsoft keys were found in self-signed certificates in majority of the scans. Self-signed certificates are implicitly not trusted by web browsers. Only a subset of the not self-signed certificates have certificates chains leading to a browser-trusted root CA.

Source	Version	Year	Group	Prime bias	Mod 4 bias	Mod 3 bias
Open-source libraries						
Botan	1.5.6, 1.11.29, 2.1.0	2006, 2016, 2017	13	11_2		
Bouncy Castle (Java)	1.53	2016	12	RS		
Bouncy Castle (Java)	1.54	2016	11	$\sqrt{2}$		
Cryptix JCE	20050328	2005	12	RS		
cryptlib	3.4.3, 3.4.3.1	2016, 2017	13	11_2		
Crypto++	5.6.0, 5.6.3, 5.6.5	2009, 2015, 2016	11	$\sqrt{2}$		
FlexiProvider	1.7p7	2014	12	RS		
GNU Crypto	2.0.1	2005	3	RS	✓	
Libgcrypt (GnuPG)	1.6.0, 1.6.5, 1.7.6	2013, 2016, 2017	13	11_2		
Libgcrypt (GnuPG)	1.6.0 FIPS, 1.6.5 FIPS	2013, 2016	13	11_2		
Libgcrypt (GnuPG)	1.7.6 FIPS	2017	11	$\sqrt{2}$		
LibTomCrypt	1.17	2015	13	11_2		
mbedTLS	2.2.1, 2.4.2	2016, 2017	12	RS		
Nettle	2.0	2010	12	RS		
Nettle	3.2, 3.3	2016	13	11_2		
OpenSSL	0.9.7, 1.0.2g, 1.0.2k, 1.1.0e	2002, 2016, 2017, 2017	7	11_2		✓
OpenSSL FIPS	2.0.12, 2.0.14	2016, 2017	13	11_2		
PGP SDK	4	2011	13	11_2		
PGP SDK	4 FIPS	2011	8	PGP		
PolarSSL	0.10.0, 1.3.9	2009, 2014	12	RS		
Putty	0.67	2017	12	RS		
SunRsaSign Provider	OpenJDK 1.8	2014	12	RS		
WolfSSL	2.0rc1, 3.9.0, 3.10.2	2011, 2016, 2017	13	11_2		
Black-box implementations						
HSM Utimaco	SecurityServer Se50		12	Uti		
HSM SafeNet	Luna SA-1700		13	11_2		
Microsoft	CNG, CryptoAPI, .NET	2016 (Windows 10)	11	$\sqrt{2}$		
YubiKey	4, 4 Nano	2015	9	Inf.		
YubiKey	NEO	2012	10	RS	✓	✓
Smartcards						
Feitian	JavaCOS A22	2015	13	11_2		
Feitian	JavaCOS A40	2016	13	11_2		
G&D	SmartCafe 3.2	2003	1	G&D	✓	
G&D	SmartCafe 4.x	2007	2	G&D	✓	✓
G&D	SmartCafe 6.0	2015	2	G&D	✓	✓
Gemalto	GCX4 72K	<2010	13	11_2		
Gemalto	GXP E64	<2010	4	Gem.		
Infineon	JTOP 80K	2012	9	Inf.		
NXP	J2A080	2011	5	NXP	✓	
NXP	J2A081	2012	5	NXP	✓	
NXP	J2D081	2014	10	RS	✓	✓
NXP	J2E145G	2013	10	RS	✓	✓
NXP	J3A081	2012	5	NXP	✓	
NXP	JCOP 41 V2.2.1	<2010	5	NXP	✓	
Oberthur	Cosmo Dual 72K	<2010	6	11_2	✓	
Oberthur	Cosmo 64	2007	13	11_2		

Table 3: List of sources with biases relevant for the analysis. There are two types of modular bias – modulo 4, due to RSA moduli being Blum integers and modulo 3, due to implementations avoiding primes p such that $p - 1$ is divisible by 3. The primes are biased due to different intervals, from which they are generated. The bias propagates to public keys. Notation: 11_2 – the primes have the two top bits set to one; RS – the primes have the top bit set to one, then short moduli are discarded; $\sqrt{2}$ – the primes are chosen from the interval $\left\lfloor \sqrt{2} \cdot 2^{\frac{n}{2}-1}, 2^{\frac{n}{2}} - 1 \right\rfloor$ (n – length of modulus). Other proprietary implementations of prime selection are: G&D – Giesecke & Devrient (G&D); Gem. – Gemalto; Inf. – Infineon; NXP – NXP; PGP – PGP SDK; Uti – Utimaco (similar to RS).