Ca' Foscari University of Venice

and

Masaryk University

Challenging RSA cryptosystem implementations

Doctoral Thesis

Matúš Nemec

Doctorate Coordinator
Prof. Riccardo Focardi

Advisor (Ca' Foscari University)
Prof. Riccardo Focardi

Cycle: 32
Matricola: 956333
SSD: INF/01 Informatica

Advisor (Masaryk University)
Prof. Vashek Matyáš

Consultant
Doc. Petr Švenda

Semester: Fall 2019
UČO: 396066

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Matúš Nemec

Advisor (Ca' Foscari University of Venice): Prof. Riccardo Focardi
Advisor (Masaryk University): Prof. Vashek Matyáš
Consultant (Masaryk University): Doc. Petr Švenda

Acknowledgements

I would like to thank Vashek Matyáš, Riccardo Focardi, Petr Švenda, and Marek Sýs for their guidance and advice. I am thankful to all my co-authors for their fantastic work, and to all the members of CRoCS and ACADIA groups for creating a welcoming atmosphere. I much appreciate the help of Nicola Miotello and Ada Nazarejová with going through the red tape. Last but not least, this thesis would not be possible without the support of my family and friends.

Abstract

The aim of our research was to study security properties of real-world deployments of the RSA cryptosystem. First we analyze RSA key generation methods in cryptographic libraries. We show a practical application of biases in RSA keys for measuring popularity of cryptographic libraries and we develop a factorization method that breaks a proprietary key generation algorithm. Later we examine published implementation issues in the TLS protocol, such as RSA padding oracles, in the wider context of the Web ecosystem.

The main conclusion of our previous research on RSA keys was that libraries leak information about their key generation algorithms through the public keys. The bias in the key is often sufficient to identify a library that likely generated the key. In this thesis, we further demonstrate a practical method for measuring representation of cryptographic libraries in large datasets of RSA public keys. We use statistical inference to approximate a share of libraries matching an observed distribution of RSA keys in an inspected dataset. Such estimate also allows us to deploy an improved key classification method that uses prior probabilities of libraries to more accurately determine the source of a key.

Next we adapt a key factorization attack to keys generated by a widely deployed proprietary cryptographic library. Our previous research hinted at issues in the algorithm by showing an unusual bias in the keys. We reveal the algorithm without the access to the source code by analyzing algebraic properties of the keys. We develop a practical factorization attack that can compute private keys corresponding to the public keys used in electronic identification documents, Trusted Platform Modules, authentication tokens, and other domains that use secure chips with this RSA key generation algorithm. Our findings were disclosed in coordination with the manufacturer of the devices to minimize the resulting security threats.

Finally, we focus on implementation issues in the TLS protocol, of which RSA key exchange is an integral part. We survey existing practical attacks that affect current usage of TLS for securing the Web. We specify conditions for the attacks as attack trees. Using measurements on popular domains, we confirm that RSA padding oracles are

currently the most common threat. We show how such security issues get amplified by the complexity of the Web ecosystem. In many cases of vulnerable websites, the issues are caused by external or related-domain hosts.

Our work helps to demonstrate how RSA, a seemingly simple and intuitive cryptosystem, requires a lot of knowledge to be implemented correctly. Unlike RSA, elliptic curve cryptography (ECC) algorithms do not require padding, and parameters can be chosen such that random strings serve as keys. ECC is more resistant to bad user configurations and provides many other benefits. We conclude that practitioners should follow the example of TLS version 1.3 and stop using RSA in favor of ECC.

Keywords

Bleichenbacher's oracle, Coppersmith's algorithm, cryptographic library, cryptographic protocol, factorization, key generation, RSA, TLS

Contents

1 Introduction

The RSA cryptosystem predates most currently used cryptographic libraries. It is often a necessary part of a new library and perhaps one of the first features to be implemented. In a survey of RSA key generation implementations [Šve+16a], we saw how the source code for RSA in community-driven open-source libraries often remained untouched over many library versions. We sought to challenge RSA cryptosystem implementations to find out whether they were the best they could be or lagging behind the constantly advancing attacks and recommendations. It was proven time after time that it is necessary to repeatedly scrutinize both old and new systems, to actively improve the security of practical applications.

Cryptographic implementations were shown to fall short of expectations set by standards, such as when RSA key generation in cryptographic libraries did not follow standards [Šve+16a], the size of the Diffie-Hellman subgroup was not validated by libraries as required [Val+17], or random number generators were not working correctly [Hen+12; Bel08; Bar+16; HFH16]. By scrutinizing not just specifications, but also practical applications, urgent problems can be discovered and addressed. To complicate the matter, without the cooperation of the authors, proprietary systems must be evaluated without the knowledge of their inner workings. This is addressed by black-box testing [Som16b] and reverse engineering [Nem+17b].

With the presence of bugs in library implementations, it is also desirable to measure impacts of resulting security vulnerabilities. In some cases, the observation cannot be made directly, but the applicability of the attack becomes known only after expending non-trivial work. Previously, we proposed a method for inferring the cryptographic library from bias in an RSA key [Šve+16a], reducing the uncertainty of the presence of a vulnerable library. However, we also identified the shortcoming of the method, which we address in this thesis.

Despite some attacks being known in theory, often system designers do not deploy countermeasures unless the feasibility of the attack is demonstrated. There are several examples in the TLS protocol suite. The RC4 stream cipher was known to have biased keystream [MS02], but it was only deprecated [Pop15] in TLS when practical attacks were

constructed [AlF+13; GPM15; VP15]. Weak cryptographic algorithms, such as the MD5 hash function and deliberately weakened RSA and Diffie-Hellman parameters were supported, until SLOTH (transcript collision attacks) [BL16a], FREAK [Beu+15] and Logjam [Adr+15], respectively, exploited them. Legacy protocol versions often remain supported beyond their usefulness, leading to attacks on the weakest part of the system [JSS15; Avi+16]. Even old attacks need to be revisited and their applicability to new systems verified. Malicious parties may have already done so and may be exploiting the weaknesses.

1.1 Problem statement

We aim to scrutinize RSA key generation methods from the perspectives of design and implementation, including an empirical analysis of keys used in real systems. We also study the resiliency of TLS protocol implementations to a variety of known attacks, amplified by the complex dependencies in the Web ecosystem. In summary, the main areas of our research are:

- *Biased RSA key generation methods* – in the literature (standards and practical implementations) there is no consensus on RSA key generation methods. As a result, alternative methods produce keys with different biases. We aim to explore the consequences of using keys from biased distributions, such as RSA key classification and cryptographic library popularity measurements.

- *Vulnerabilities in RSA key generation* – we aim to identify design choices of a selected RSA key generation method to evaluate its security properties.

- *TLS protocol vulnerabilities* – many diverse attacks against implementations of the TLS protocol were published. It can be challenging to understand how they can be mounted and mitigated. Their consequences are not easily evaluated for a specific target in the intricate system of servers running the Web. We aim to ease these tasks and make the results understandable to a wider audience.

1.2 Contributions

We discuss the specific contributions to the research questions.

1.2.1 Biased RSA key generation methods

Accurate library popularity measurements are not readily available. They serve an important role for precise RSA key origin classification and for measuring impacts of security vulnerabilities in cryptographic libraries. We created a practical tool capable of making such measurements and able to evaluate the trends in library popularity in historical datasets.

Previous solutions that relied on indirect measurements based on proxy information were not backed by rigorous analysis. Alternative methods [Šve+16a] have known limitations. We propose a solution that uses statistical inference to approximate a share of libraries matching an observed distribution of RSA keys in an inspected dataset. The method extends our previous work [Šve+16a]. The main difference is the fact that datasets of keys are processed as a whole. The original solution considered keys individually and did not take advantage of the "big picture".

We collected large datasets of public RSA keys from various sources and applied our methods to answer questions pertaining to the popularity of cryptographic libraries on the Internet. We evaluated the trends over time and demonstrated differences across application domains.

The results were published as [Nem+17a] and are presented in Chapter 3.

1.2.2 Vulnerabilities in RSA key generation

The RSA key generation algorithm used in libraries of Infineon Technologies exhibited remarkable statistical properties, as we noted in [Šve+16a; Šve+16b]. It showed entropy loss and hinted at a possible security vulnerability. The details of the key generation algorithm were not published by the manufacturer since neither the source code nor the object code was made available.

We analyzed a large number of public and private RSA keys from Infineon smartcards using statistical and algebraic methods, in order to reveal the key generation method. We were able to correctly determine the algorithm and precisely measure the entropy loss resulting from its unsound efficiency improvements. We developed and optimized a practical factorization method that reveals the private key given the public key. An efficient fingerprinting technique was used to detect affected RSA keys in a variety of domains. Millions of electronic identity documents used in several countries were affected, as well as Trusted Platform Modules in multiple laptop models. Other applications that used the flawed library to generate RSA keys used for electronic signatures were also threatened.

The results were published as [Nem+17b] and are presented in Chapter 4. The vulnerability received a lot of media attention [Goo17b; Ley17b; Kha17], especially in connection with electronic identity documents [Goo17a; Ley17a; Vah17].

1.2.3 TLS protocol vulnerabilities

We systematically described the conditions and consequences of practical attacks against the TLS protocol. We streamlined the analysis of whether they can be mounted and mitigated on real targets in a wider scope of the Web ecosystem. We performed a security assessment of a selected set of popular websites. We presented the results as specific threats to essential aspects of the security of the sites.

Our results showed that the most commonly applicable attacks are variants of the RSA padding oracle. Their consequences range from threatening the confidentiality of past recorded messages to providing an attacker with active Man in the Middle capabilities. The vulnerabilities are due to implementation errors. It has been argued that the underlying cause is the poor choice of the padding algorithm and complex collection of countermeasures that all need to be implemented correctly. The findings help to reinforce our experience that implementations are not up to date with best practices.

We increased the scope of our study from a typical list of most popular websites also to the supporting infrastructure. We were able to find much more vulnerable servers, as secondary systems often receive less attention for maintenance. Most importantly, we were able

to demonstrate how the flaws get amplified by systems that rely on these vulnerable or misconfigured servers.

The results were published as [Cal+19] and are presented in Chapter 5. The research was also covered by media [New19].

1.3 Structure of the thesis

The thesis is further divided as follows:

- Chapter 2 describes our initial research on RSA key generation methods, leading to two major research areas of the thesis. It represents the state of the art before we made our contributions to the topics. More specialized related work is discussed individually in the chapters that cover our concrete research questions.
- Chapter 3 presents a practical application for the existence of biased RSA key generation methods. We design and perform measurements of the popularity of cryptographic libraries on the Internet, over time, and across different application domains.
- Chapter 4 is dedicated to the flawed RSA key generation algorithm found in a cryptographic library of a manufacturer of secure chips. We describe our attack, its optimization, and performance evaluation. We evaluate various domains for the presence of vulnerable keys.
- Chapter 5 details our study of the feasibility and consequences of practical TLS attacks. We demonstrate how a relatively few exploitable vulnerabilities get amplified by the complexity of the Web ecosystem.
- Chapter 6 concludes our research on the flaws and shortcomings of RSA implementations. We summarize the main outcomes and future work.
- Appendix A lists the publications of the author of the thesis, including the specific contributions made by the author.

2 State of the art

The topic of this thesis was inspired by the 2016 paper *The Million-Key Question – Investigating the Origins of RSA Public Keys* [Šve+16a] co-authored by the author of the thesis. In this chapter, we summarize its results of surveying implementations of RSA key generation. The core idea of the paper – to explore biases in RSA keys in practice and leverage them for RSA key classification – was unique. Hence the single paper covers state of the art for a large part of our research presented herein. The outcomes are very relevant for Chapters 3 and 4. The paper also helps to demonstrate how complicated specifications contribute to problems such as those studied in Chapter 5.

Chapters detailing our particular research questions provide more specialized summaries of the more closely related work. Namely, in Chapter 3, we dive deeper into the RSA key classification and discuss alternative ways of estimating the popularity of libraries in Section 3.5. For Chapter 4, we discuss RSA key generation methods and factorization attacks on RSA in Section 4.6. We focus on TLS attacks throughout Chapter 5, presenting both a survey of attacks in Section 5.3 and references to related studies and measurements both for TLS and Web security in Section 5.8.

2.1 Survey of RSA key generation methods

To use the RSA algorithm, one must generate a key:

1. Select two distinct large primes[1] p and q.
2. Compute $N = p * q$ and $\varphi(N) = (p-1) * (q-1)$.
3. Choose a public exponent[2] $e < \varphi(N)$, e coprime to $\varphi(N)$.
4. Compute the private exponent d as $e^{-1} \bmod \varphi(N)$.

1. Generated randomly, but possibly constructed to achieve certain required properties.
2. Usually with a low Hamming weight for faster encryption.

The pair (e, N) is the public key; either (d, N) serves as the secret private key, or (p, q) can be used ((d, N) can be calculated from (p, q, e) and vice versa).

We showed in [Šve+16a] that authors of different cryptographic libraries adopt various methods for RSA key generation. The situation arose from the existence of different competing standards and publications on the topic, as well as from varying limitations and requirements placed on the libraries and similar products. Even subtle differences in the algorithms can introduce small bias into the distributions of the produced keys. We identified the most common sources of the biases that can be detected in public or private keys.

Previous surveys of RSA prime generation algorithms were published [LN11; LN14; JP06]. The crucial difference of our work [Šve+16a] was that we focused on the actual outputs of the implementations. Some libraries followed standardized algorithms; many did not include any specification or reference for their algorithms. In addition, coding mistakes can create further discrepancies with the original algorithm. A more recent survey [Alb+18] focused on primality testing in implementations. It revealed how a few publications could shape the choices of many independent implementations. Primality testing is often described with the assumption that tested values are random, and the same assumption got adopted in many cryptographic libraries. However, in practical applications, the libraries may receive adversarial inputs crafted by attackers to fool the primality test using specially constructed composite numbers.

Our paper also shows how to use bias extracted from a specific key to classify the key as originating from a concrete class of RSA key generation implementations. In the best case, a single library can be identified. On the other hand, large number of libraries behave similarly, hence they belong to the same class. Even though the chosen classification method is simple, a single key is correctly classified on the first try with average accuracy over 40% for 13 classes.

This result was extended to large datasets of public RSA keys. However, the produced statistics describing the collections of public RSA keys from the Internet were skewed, due to keys being handled individually. We developed a new method that considers the big picture. It was published in [Nem+17a] and described in Chapter 3.

Our survey paper was also the first to point out extensive biases in keys generated by Infineon smartcards. By studying the biases more precisely in subsequent work, we were able to reverse engineer the key generation algorithm and devise a practical RSA key factorization method as seen in Chapter 4. The results were published as [Nem+17b].

2.2 Bias in RSA primes and keys

Bias in RSA primes was previously used to identify the source of RSA keys factored using an efficient batch variant of the greatest common divisor (GCD) algorithm [Hen+12; Mir12]. Our previous work [Šve+16a] identified more types of bias found in RSA keys.

2.2.1 RSA prime types

According to the survey [Šve+16a], implementations most commonly generate probable, provable, and strong primes. Papers such as [LN11; JP06] imply an additional category of constructed primes.

Some categories can be distinguished based on the factorization of $p-1$ and $p+1$, p being one of the RSA primes. For every strong prime p, both $p-1$ and $p+1$ contain a factor of a certain length (or the length is from a specific interval). Provable primes p have such factor in $p-1$, but not in the factorization of $p+1$. Probable primes may show some bias in the factorizations, yet there are no factors of large fixed length in either $p-1$ or $p+1$.

The paper does not explicitly give a method to tell apart probable primes generated randomly and by an incremental search. In the former case, random numbers are generated until a prime is found. In the latter case, a single random value is generated and incremented until a prime is found. Large primes are not evenly spaced. Incremental search biases the distribution of primes toward primes that are preceded by a larger gap made of composite numbers. Generating any value from a specific gap results in the same prime, hence gaps with more values have a higher probability of being selected. It follows that the distribution of the distance to the previous prime carries different types of bias for these two methods.

2.2.2 RSA keypairs

The RSA modulus is a product of two primes. The libraries usually generate a modulus of a precise length supplied by the user, and the primes have half the bit length of the modulus. A product of two k-bit numbers has either $2k$ or $2k - 1$ bits. It is possible to generate two k-bit numbers and discard one or both of them in case their product is too short. It is more common to generate the primes from a smaller interval or to modify the high bits of the candidate values to ensure that their product has the correct length. The choice of interval affects the distribution of the most significant bytes of primes and resulting RSA moduli.

2.2.3 Modular bias

All primes are odd, but some libraries also make values biased modulo larger numbers. A specific case is the OpenSSL library, which does not allow RSA primes equal to one modulo any prime from 3 to 17863. This observation can serve as a fingerprint of the private key, as the probability that two random primes have this property is quite low. It was used to identify keys originating from OpenSSL in a study of keys factorable by the Batch GCD method due to shared primes [Mir12].

In the extended technical report [Šve+16b] of our survey [Šve+16a], we correctly identified further bias in Infineon primes and moduli. The remainders modulo certain small primes were only from specific subgroups of residue classes and did not represent all residue classes. It was observed that the values were almost uniformly distributed modulo all other tested primes. In this thesis and [Nem+17b], we extended this analysis to composite moduli and were able to describe the bias modulo a product of several consecutive primes. All primes originated from a single subgroup generated by a single generator.

2.3 Pseudo-random number generator failures

The pseudo-random number generator (PRNG) provides random bits for the key generation. In the case of probable primes, it is usual that the random bits are directly used to construct the candidate value. Hence we can test the bytes of the prime instead of the PRNG output.

We skip the bottom bytes that get incremented in incremental search and the top bytes that are modified to fit the intervals. In [Šve+16a], we performed a simple non-overlapping serial test on two consecutive bytes, finding one case of apparent bias.

One family of smartcards occasionally produced keys that shared primes that make the keys easy to factor, if they share a prime [Hen+12]. The reason was an unchecked failure of the PRNG that provided zero-value bytes. The top bits were modified, and the value was incremented until a prime was found. It is not necessary to have a fully predictable PRNG output. Even a partial or approximate knowledge of a prime can be used to factor keys with a variant of Coppersmith's attack [Cop96a; Ber+13].

In many cases of constructed primes, we do not have direct access to the random sequence. The resulting prime is constructed from partial values. For example, if the library uses strong primes, we would have to factor $p - 1$ and $p + 1$ and correctly identify which factors were generated as the auxiliary primes.

3 Measuring popularity of cryptographic libraries

In 2016, we surveyed RSA key generation methods used in practice [Šve+16a]. We demonstrated that public RSA keys carry enough bias to classify them to their originating cryptographic library with much higher success than for random guessing. As an additional practical demonstration, we processed millions of keys from public sources and used our method to attribute them to a probable source. However, when we summarized the most likely sources of all keys, some objectively unlikely libraries were represented, such as cryptographic smartcards in the domain of TLS servers.

In this chapter, we explain and fix the imprecise measurements. In summary, the problem lies in looking at keys individually instead of considering them all at once. The classification of an individual key always returns the library that produces such key with the highest likelihood. However, the likelihood would be different for a group of two, three, or more keys. We must consider that a library may contribute anything from zero to millions of keys used on the Internet.

Individual instances of the same key generation algorithm running independently on thousands of machines produce results very similar to what we collected by running it on a single computer repeatedly. Hence if we look at all the keys generated collectively by millions of computers, all of them using one of a few cryptographic libraries, we can observe a mixture of the distributions that we collected from the same libraries locally. The main principle of our new approach is to discover the parameters of the mixture – how much of the whole is contributed by each library. This, in turn, gives us an approximation of the popularity of cryptographic libraries used on the Internet. Based on our experimental measurements and comparisons with other indicators on the usage of libraries, our new method is much closer to the truth than before.

The results in this chapter were published in [Nem+17a].

3.1 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 [Laz+14] 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 [BB05] or AES secret keys [Ber05] remotely from a targeted web server and generation of vulnerable keys by a weak or a malfunctioning random generator [Bel08; Hen+12]. 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 [Šve+16a]. 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 [Šve+16a] 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 work 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 chapter is organized as follows: Section 3.2 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 3.3 explains the details of the library popularity measurement method and discusses the accuracy. Section 3.4 applies our method to large current and archived datasets to measure the popularity of libraries in time and discusses the observed results. Section 3.5 provides a review of related work. We list some additional results in Section 3.6. The chapter is concluded in Section 3.7.

3.2 Method overview

The authors of [Šve+16a] 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. One might expect that cryptographic keys would be chosen from a uniform distribution, to help prevent brute-force attacks. However, most of the libraries examined by [Šve+16a] produced primes with an unevenly distributed most significant byte. As a result, the distribution of the modulus was also non-uniform.

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 [Šve+16a]. 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.

3.2.1 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.

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 [Šve+16a]. 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 3.1, illustrated by the distributions of mask values for OpenSSL. When compared to [Šve+16a], 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 3.3 in Section 3.6.

3.2.2 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 [Šve+16a] used a highly representative sample of cryptographic libraries in what were then the most recent available versions. For this

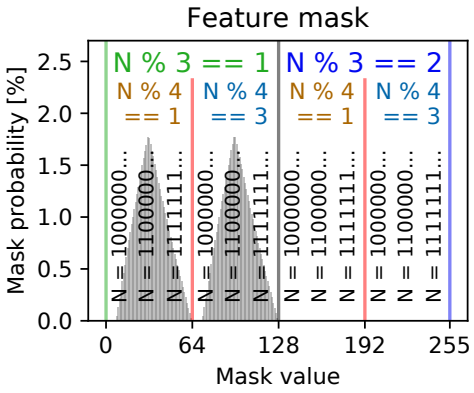

Figure 3.1: Features extracted from a public RSA modulus. The plot shows the probability that the given library generates a key with the corresponding mask value. The moduli generated by OpenSSL are always equal to one modulo three ($N \,\%\, 3 = 1$), but they are uniformly distributed modulo four ($N \,\%\, 4 = 1$ 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.

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 cluster-

ing 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 [Šve+16a]. The result of the clustering analysis is visualized in Figure 3.2 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.

3.2.3 Dataset classification – original approach

The main focus of the authors of [Šve+16a] 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:

$$P(G|K) = \frac{P(K|G)P(G)}{P(K)}, \tag{3.1}$$

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.

3.2.4 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

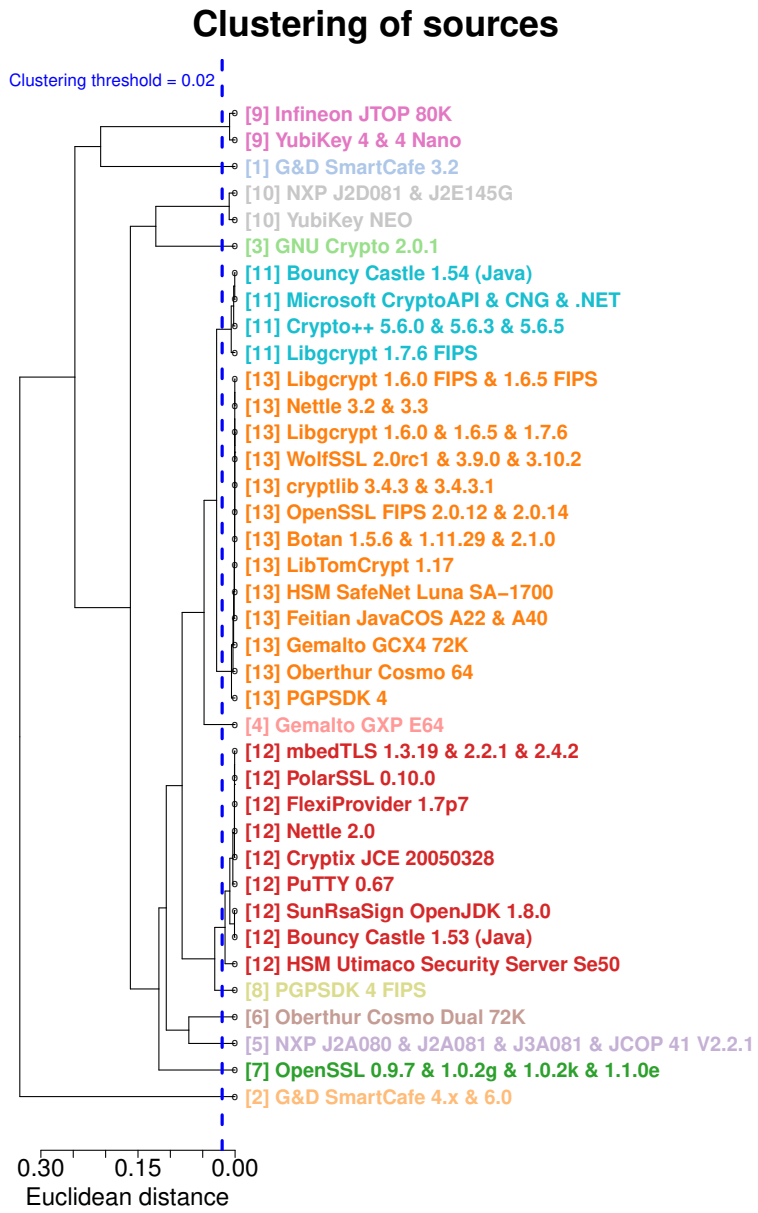

Figure 3.2: The result of the clustering analysis visualized as a dendrogram. Clusters are created based on the Euclidean distance, with a separation threshold of 0.02 (blue dashed line), similarly as the approach of [Šve+16a]. The group numbers are listed in brackets next to the source name.

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) [LH95]. 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 3.3.

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 [Šve+16a] 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 3.3.4.

3.2.5 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.

3.3 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.

3.3.1 Model

We assume there are m groups of sources, created by clustering analysis (Section 3.2.2) 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 3.2.1) is given by:

$$P(K) = \sum_{j=1}^{m} P(G_j)P(K|G_j), \tag{3.2}$$

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.

3.3.2 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 3.3 – 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 3.2. In our case, the system has 256 equations with 13 unknowns $P(G_j)$. Since both the distribution of real world keys and the reference distributions are empirical, a precise solution may not exist, due to the presence of noise (see Section 3.3.4).

We chose the linear least squares method constrained to non-negative coefficients (non-negative least squares fit or NNLSF) implemented in Java [Kam13] based on the algorithm by Lawson and Hanson [LH95] 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 3.2 with the estimated group probability $\hat{P}(G_j)$.

3.3.3 Key classification

We classify the keys according to their origin using the Bayes' rule (Section 3.2.3). When compared to the approach of [Šve+16a], 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.

3.3.4 Evaluation of accuracy

We are interested both in the accuracy of the prior probability estimation (given as the expected error in the estimation, Table 3.1) and in

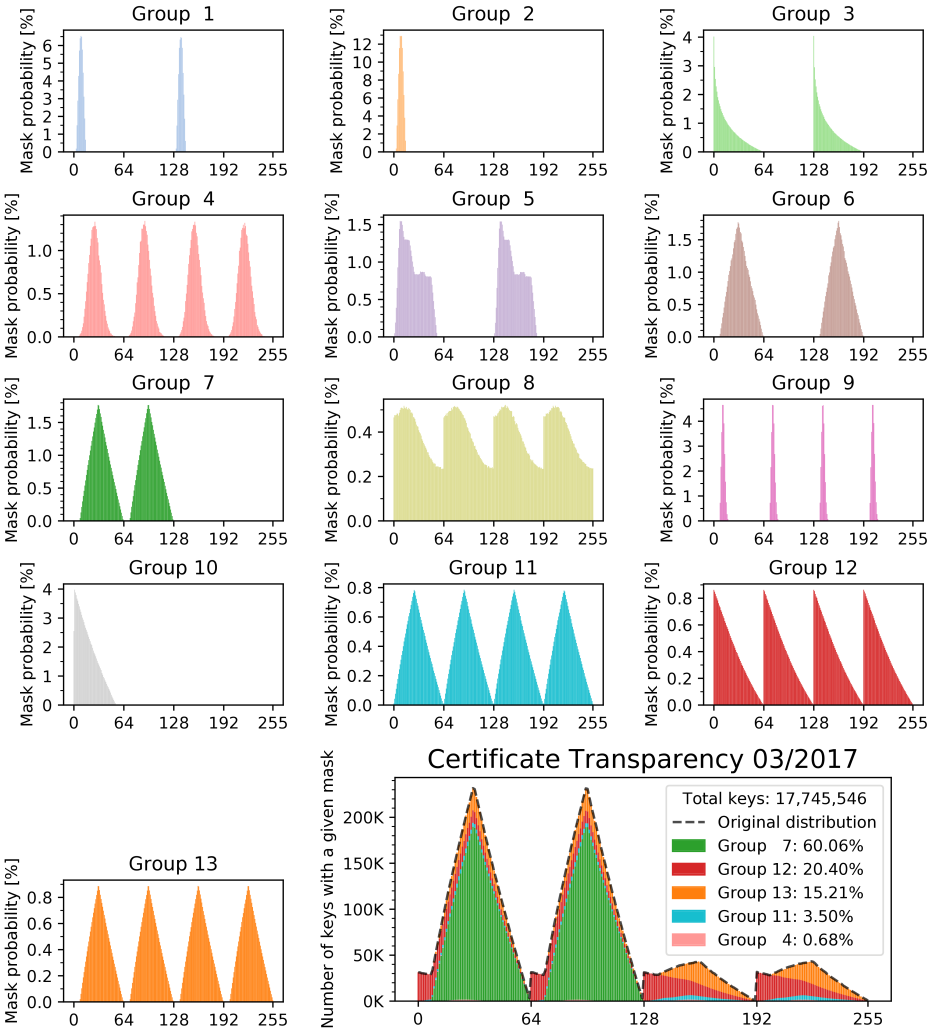

Figure 3.3: The reference distributions of the key mask from each library are used to compute the probability with which the given library contributes to the overall distribution of keys in the measured sample. The distribution of keys submitted to Certificate Transparency logs during March 2017 likely 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).

the average correctness of the overall classification process (given as the proportion of keys that were correctly classified, Table 3.2). For the measurement, we repeatedly simulate large datasets according to different distributions, add noise and perform our method.

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).

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 3.1 and 3.2) 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.

Simulation scenarios

We considered several distributions of prior probability library usage:

Evenly distributed probabilities match the approach in [Šve+16a], 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 3.2.1).

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.

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 3.1.

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 $\pm 3\ pp$. The averages in Table 3.1 are given for 100 experiments, each simulating one million keys. We considered distinct scenarios (Section 3.3.4) and levels of systematic noise (Section 3.3.4).

	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 3.1: 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.

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 3.2 are given in percents.

Tables 3.1 and 3.2 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 [Šve+16a], 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.

3.3.5 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

	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 3.2: 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.

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 3.4, 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.

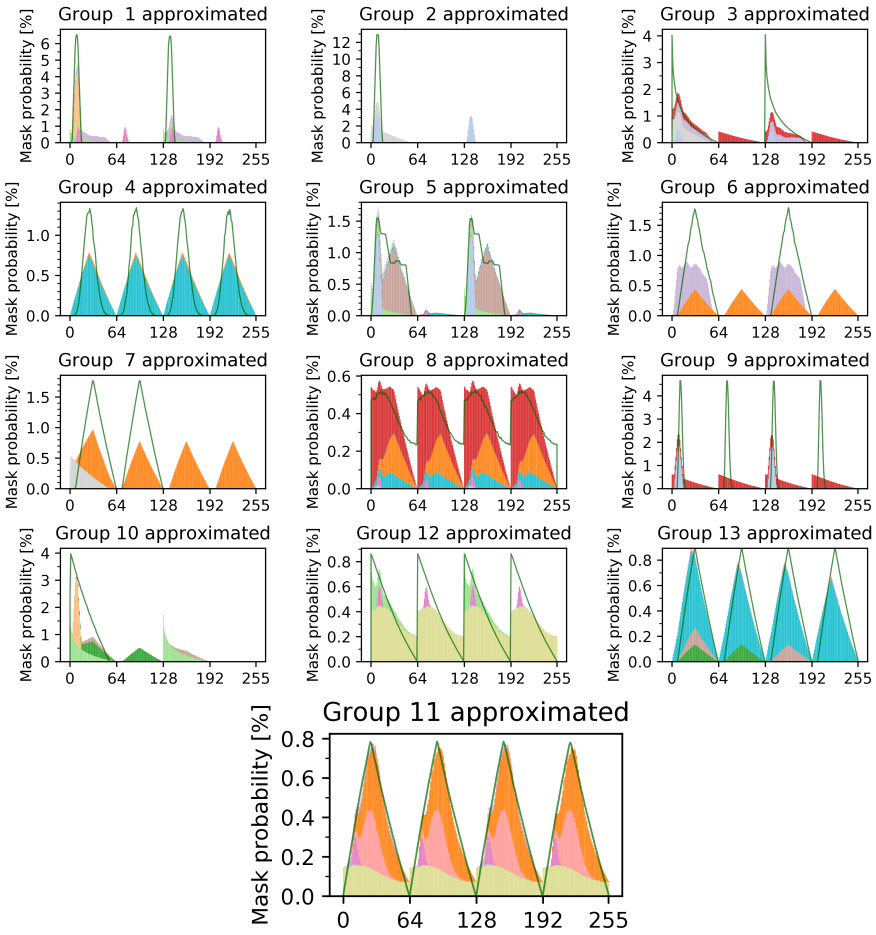

Figure 3.4: Some distributions may be interchanged for a mix of other distributions. We used our method to approximate each of the 13 distributions using only the remaining 12 distributions. The graphs show the original distribution as a green outline and the combination of libraries that minimizes the 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.

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.

3.4 Results on relevant datasets

Rough estimates of popularity for some cryptographic libraries were provided in [Šve+16a] 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.

3.4.1 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.

Censys TLS scan

Censys [Dur+15a] performs a full IPv4 address space scan of TCP port 443 on a weekly basis [Cen15b]. 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 [Col15]. 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.

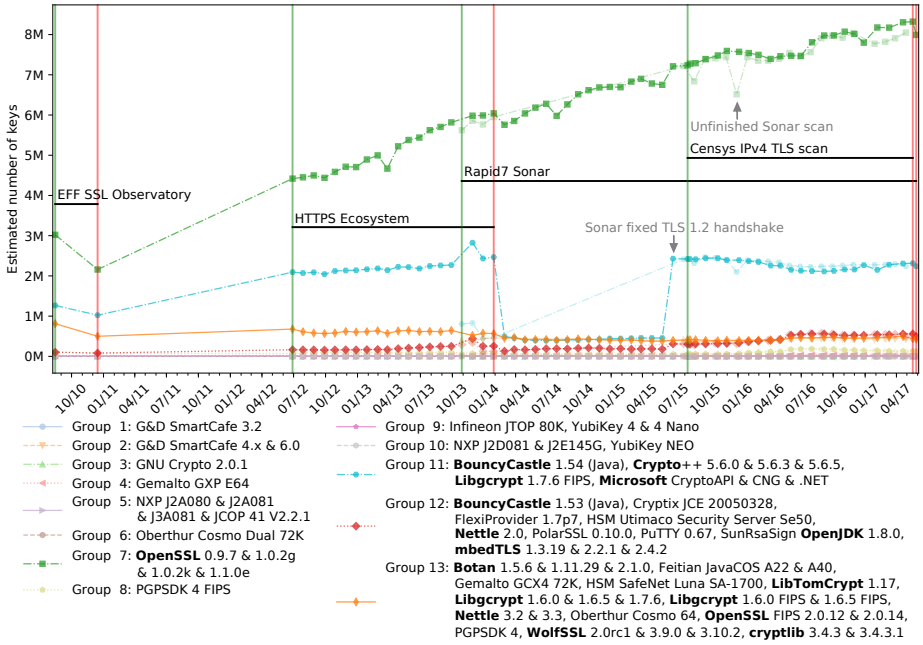

Figure 3.5: The combined results from scans of TLS services in the whole IPv4 space as provided by four independent datasets, given with one-month granularity. An absolute number of unique keys as attributed to different groups by our method are shown. The sudden "jump" for Group 11 (Microsoft libraries) 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.

Censys Alexa 1M

The dataset [Cen15a] 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.

Rapid7 Sonar SSL

Project Sonar [Rap15] 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.

HTTPS Certificate Ecosystem

IPv4 TLS scanning dataset [Dur+13] 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.

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).

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

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 subkeys 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 [KS15] 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.

3.4.2 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 3.4.1.

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 3.5 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.

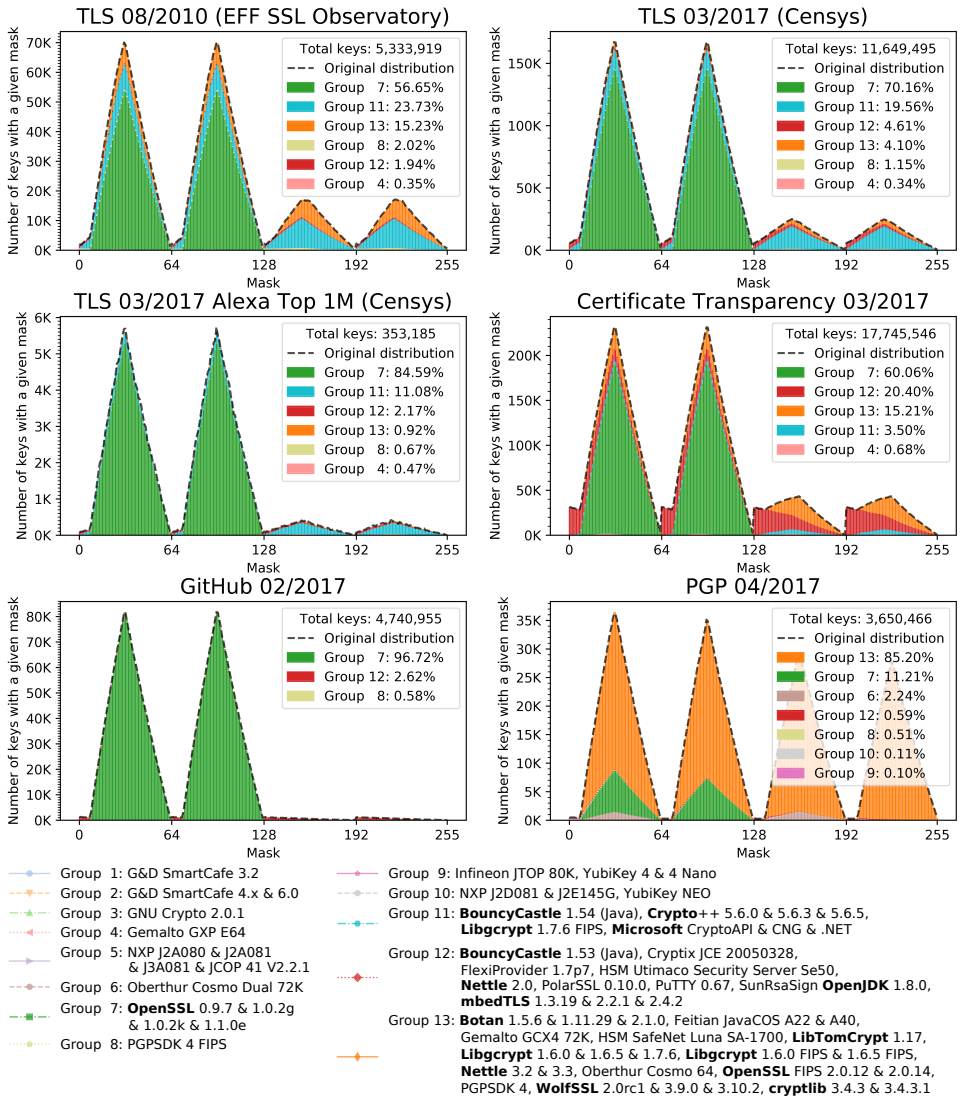

Figure 3.6: Library share in different usage domains. The sources responsible for at least 0.1% of all keys in a particular scan of the domain are listed in the legend. OpenSSL library dominates in all domains, except for the PGP dataset.

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 3.3.5.

3.4.3 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 3.6.

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%.

3.4.4 TLS to CT comparison

According to a survey based on IPv4 scans, Certificate Transparency (CT) and a large set of active domain names [Van+16], 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) 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 [Gus+17], 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 3.6. 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.

3.4.5 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 October 2016 and May 2017, limited to certificates issued by Let's Encrypt CA, as shown in Figure 3.7. 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.

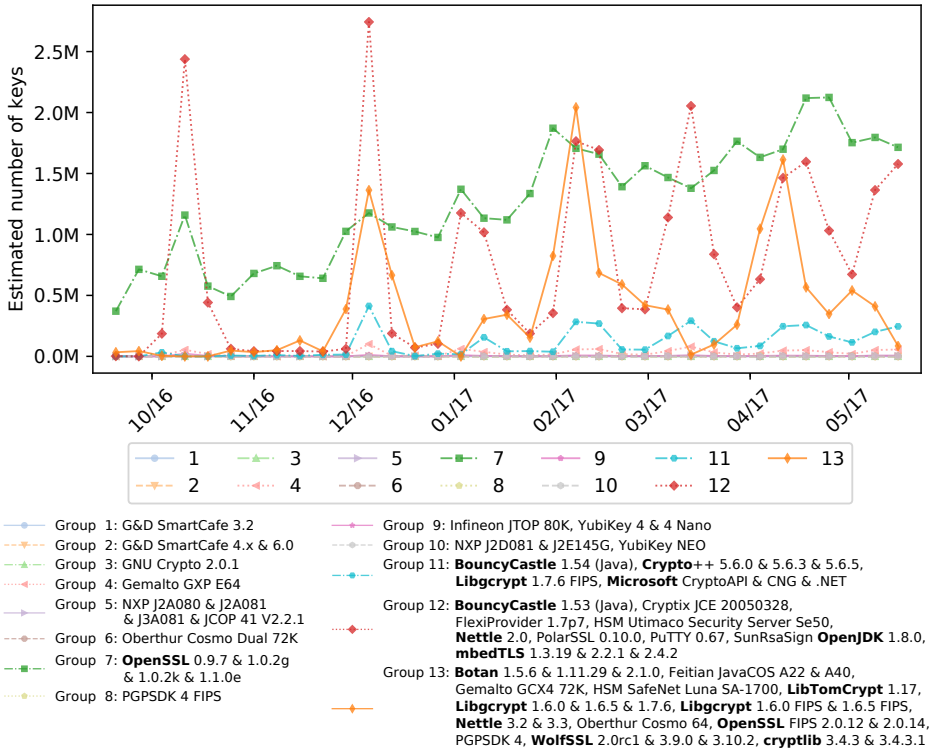

Figure 3.7: The number of keys from distinct groups added to CT weekly, found in certificates issued by Let's Encrypt CA.

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.

3.5 Related work

Only very few prior publications are concerned with the identification of the library responsible for generating an RSA key. Except for [Šve+16a] (the work we directly improve on), the task was done by [Mir12], 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 [Hen+12; HFH16; Bel08]. 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 [Net17b; Lyo17]. Indeed, the estimates on the number of servers running Microsoft OS published by [Net17a] 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 [Alb+16] 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 [GB17]. 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 li-

braries used to validate SSL certificates in non-browser client software were surveyed in [Geo+12].

A direct identification of software packages running on other cores in a cloud environment based on cache side-channels was demonstrated by [Ira+15; Inc+16]. 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 [BE10], followed by analyses of both valid certificates [CO13; Dur+13; Dur+14; DBH14; Fel+17] (the majority of papers) as well as invalid ones [Chu+16]. 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 [Van+16]. 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 [Dur+15a; ICS17], 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 [Reb15; Bar+16] with a focus on the algorithms, key lengths, and presence of weak keys, detecting keys generated from OpenSSL with insufficient entropy.

3.6 Additional results

Table 3.3 shows the sources considered in the analysis, together with the relevant biases. 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 $\left[\sqrt{2} \cdot 2^{\frac{n}{2}-1}, 2^{\frac{n}{2}} - 1\right]$ (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).

Figure 3.8 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 [CO13; Dur+13; Dur+14; DBH14] compared various properties of certificates. Valid and invalid certificates were compared by [Chu+16], 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 3.9 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 3.4.2, 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.

3.7 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 [Šve+16a].

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.

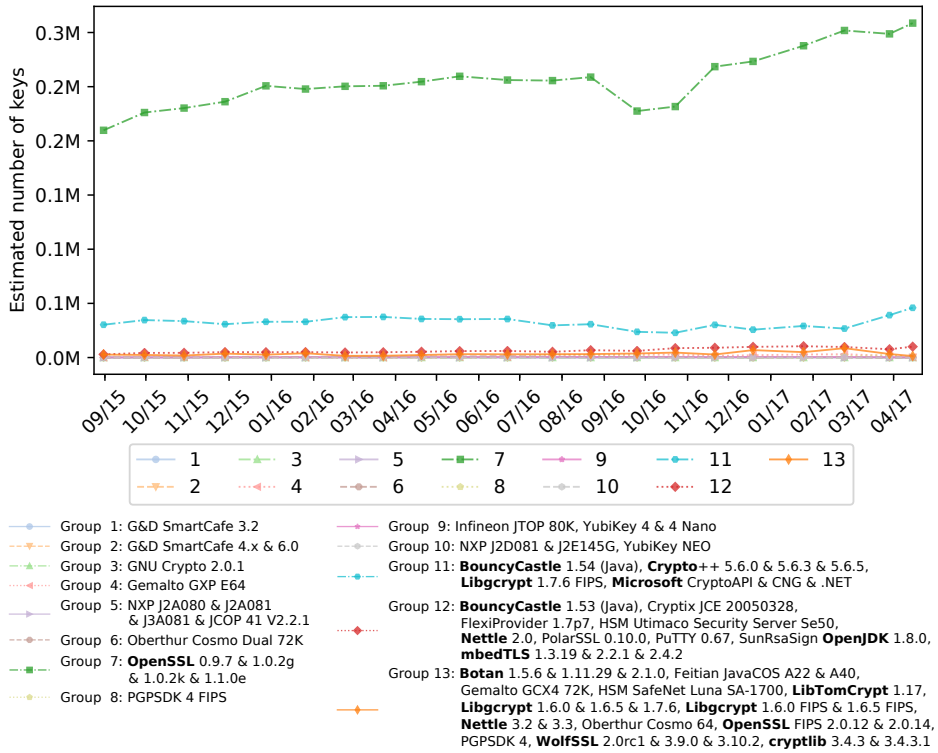

Figure 3.8: More domains from the Alexa Top 1M list use OpenSSL (Group 7) now than in 2015. Note that the number of keys does not sum to 1M already in the original dataset collected by Censys. Some websites do not support HTTPS [Fel+17] or the specific cipher-suite used by the Censys scanner.

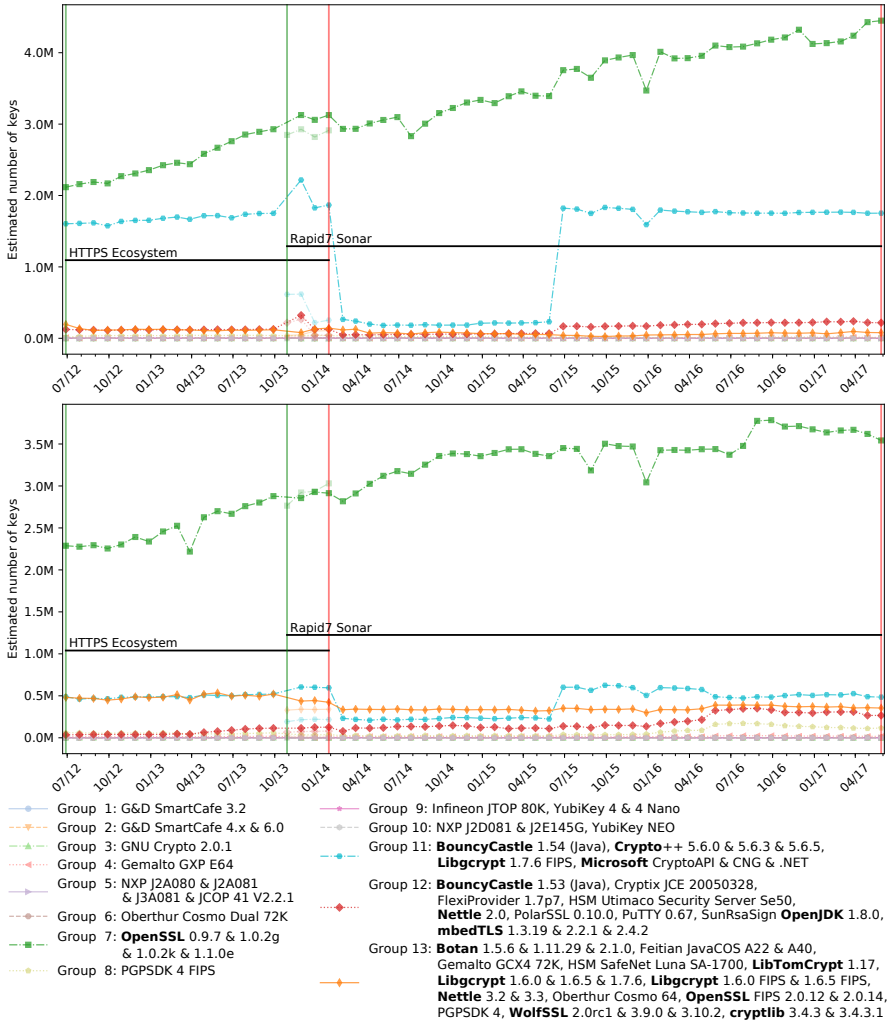

Figure 3.9: Comparison of library popularity for keys coming from certificates signed by a third party (top) and self-signed certificates (bottom). Self-signed certificates are dominated by OpenSSL. More than 50% of OpenSSL keys observed in 2012 were found in self-signed certificates. For OpenSSL, the 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	Bias		
				Prime	%4	%3
Open-source libraries						
Botan	1.5.6, 1.11.29, 2.1.0	2006, '16, '17	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, '17	13	11_2		
Crypto++	5.6.0, 5.6.3, 5.6.5	2009, '15, '16	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, '16, '17	13	11_2		
Libgcrypt (GnuPG)	1.6.0 FIPS, 1.6.5 FIPS	2013, '16	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, '17	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, '16, '17, '17	7	11_2		✓
OpenSSL FIPS	2.0.12, 2.0.14	2016, '17	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, '14	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, '16, '17	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 (Win 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 3.3: List of sources with biases relevant for the analysis.

4 Factorization of widely used RSA moduli

In general, slight biases in public RSA keys do not have adverse effects on the security of the keys. We demonstrated that an attacker could learn what library was used to generate a given key [Šve+16a] or a curious statistician could measure the popularity of cryptographic libraries on the Internet [Nem+17a] (Chapter 3). Still, such results are far from practically affecting the confidentiality or integrity of systems that use RSA.

However, we also identified a case of severely biased RSA keys generated by a smartcard [Šve+16a; Šve+16b]. In this chapter, we describe how we used these indicators of a cryptographic weakness to reveal and break the proprietary RSA key generation algorithm. Our attack is based on the factorization of partially known RSA keys, a technique closely related to the hidden number problem. We attributed the observed biases to a severe entropy loss caused by the deployed optimized algorithm. For a vulnerable public key, it is feasible to search the whole space of random values and compute the redundant information for a full factorization of an RSA modulus, including practical key sizes, such as 2048 bits. Furthermore, vulnerable keys can be identified in just microseconds due to a unique fingerprint.

Our findings had numerous practical consequences. Millions of electronic identity documents had to be modified or reissued, a wide range of Trusted Platform Modules required firmware updates, and users of authentication tokens had to generate new keys. The ACM recognized our contributions and awarded us with the Real-World Impact Award at the CCS 2017 conference. More than two years after the discovery, various applications of the secure chips running the vulnerable library remain insecure.

The results in this chapter were published in [Nem+17b].

4.1 Introduction

RSA [RSA78] is a widespread algorithm for asymmetric cryptography used for digital signatures and message encryption. RSA security is based on the integer factorization problem, which is believed to be computationally infeasible or at least extremely difficult for sufficiently

large security parameters – the size of the private primes and the resulting public modulus N. As of 2017, the most common length of the modulus N is 2048 bits, with shorter key lengths such as 1024 bits still used in practice (although not recommend anymore) and longer lengths like 4096 bits becoming increasingly common. As the private part of the key is a very sensitive item, a user may use secure hardware such as a cryptographic smartcard to securely store and use the private key value.

Successful attacks against RSA based on integer factorization (finding the private primes p and q from the public modulus N) enable the attacker to impersonate the key owner and decrypt private messages. The keys used by secure hardware are of special interest due to the generally higher value of the information protected – e.g., securing payment transactions.

RSA requires two large random primes p and q, that can be obtained by generating a random candidate number (usually with half of the bits of N) and then testing it for primality. If the candidate is found to be composite, the process is repeated with a different candidate number.

However, there are at least three reasons to *construct* a candidate number from several smaller (randomly) generated components instead of generating it randomly: 1) an improved resistance against certain factorization methods, such as Pollard's $p - 1$ method [Pol74]; 2) certification requirements such as the NIST FIPS 140-2 standard, which mandates that for all primes p, the values of $p - 1$ and $p + 1$ have at least one large (101-bit or larger) factor each; and 3) speedup of keypair generation, since testing random candidate values for primality is time consuming, especially on restricted devices like smartcards.

Yet, constructed primes may bring new problems as demonstrated in our work. In the past, practical attacks against RSA exploited the use of insecurely short key lengths susceptible to factorization via NFS [Pol93] (e.g., 512-bit, still found on the Internet [Hen+12]); faulty or weak random number generators producing partially predictable primes, as in the electronic IDs of Taiwanese citizens [Ber+13]; software bugs causing primes to be generated from an insufficiently large space, as in the Debian RNG flaw [Bel08]; or seeding with insufficient entropy, leading to multiple keypairs sharing a prime [Hen+12]. The knowledge or recovery of all bits of a private key is not always required

for a successful attack thanks to the powerful technique proposed by Coppersmith [Cop96a]. If at least one half of the bits of one of the primes is known, the remaining bits can be computationally recovered. Then, even otherwise secure designs can be attacked by various side-channel and implementation-based attacks or by introducing faults into the computation.

Only on very rare occasions is an attacker potentially able to recover the private primes of a chosen key of a seemingly sufficient bit length, without physical access to the target device or a large amount of side-channel information. One notable attack that comes close is a simple GCD computation [BHL12], which can quickly factorize a collection of moduli, but only if they happen to share a common prime, making the attack less likely to succeed on a single targeted keypair. The cause of such vulnerability is typically insufficient entropy during the keypair generation, as demonstrated for a large number of TLS and SSH keys [Hen+12; HFH16], therefore requiring multiple public keys to be created with the same malfunctioning implementation of a random number generator.

We present our attack against keys generated in cryptographic smartcards of *Infineon Technologies AG* (further denoted as *Manufacturer*), and our attack is *not* based on any weakness in a random bit generator or any additional side-channel information. Instead, the attack utilizes the specific structure of the primes as generated by *Manufacturer*'s on-chip cryptographic library (further denoted as *RSALib*[1]). We had access neither to the *RSALib*'s source code nor to the object code (since it is stored only in the secure on-chip memory and is not extractable), and the whole analysis was performed solely using RSA keys generated and exported from the *Manufacturer*'s cards and tokens.

Contributions. In short, our work has the following contributions:

1. **Recovery of the internal structure of the primes:** We identify the structure of RSA primes as produced by a black-box cryptographic library by a manufacturer of widely used cryptographic smartcards. The structure was recovered solely from our obser-

1. Likely RSA v1.02.013 library and later revisions.

vations of statistical properties of large number of private keys generated in accordance with the specification of the product.

2. **Practical factorization:** We propose and implement a technique for the factorization of such RSA keys, with lengths including 1024 and 2048 bits, using our derivation of the methods by Coppersmith and Howgrave-Graham.

3. **Fast detection algorithm:** We design a very fast algorithm to verify whether a particular key originates from the inspected library based on the properties of the public modulus. The implementation was released[2] to allow users to check their own keys.

4. **Analysis of impacted domains:** We analyze multiple usage domains (TLS, PGP, eID, authentication tokens, software signing, etc.) for the prevalence of vulnerable keys and discuss the impact of key factorization.

The specific structure of the primes as generated by *RSALib* (most likely introduced to speed up prime generation) allows us to quickly identify keys generated by the library using only the public modulus (regardless of the length of the key) and to practically factorize RSA keys with various key lengths up to 2048 bits. The factorization method uses knowledge of the specific structure of such primes to apply our derivation of Coppersmith's method. Furthermore, we devised an alternative representation of the primes in question to make the attack computationally feasible on consumer hardware.

The impact is significant due to *Manufacturer* being one of the top three secure integrated circuit (IC) producers. Furthermore, the weakness lies in an on-device software library; hence, it is not limited just to a particular range of physical devices. The weakness can be traced back to at least the year 2012, increasing the number of affected domains. We assessed the impact in a several important real-world usage scenarios and made some recommendations for mitigation.

The fingerprinting method is fast, requiring just microseconds to run on a modulus. We successfully used the fingerprinting technique

2. Full details and a tool for the detection of vulnerable keys can be found at https://crocs.fi.muni.cz/papers/rsa_ccs17.

on large datasets of certificates, such as those submitted to Certificate Transparency logs, collected in Internet-wide TLS scans and stored on public PGP keyservers. This led to a discovery of thousands of keys in the wild with primes of the form in question. Our method has negligible false negative and false positive rates (observed as zero), as guaranteed by the very rare properties.

Where datasets with public RSA keys were not available (e.g., Trusted Platform Modules or EMV payment cards), we collected sample keys from different devices to get an idea about the typical key lengths used for the domain and to estimate the prevalence of devices producing potentially vulnerable keys.

Although the *RSALib* is not automatically shipped with every chip, the developers are motivated to deploy it in order to benefit from ready-to-use higher-level functions (such as the *RsaKeyGen* method in question) and to get an implementation designed with protections against side-channel and fault induction attacks in mind. However, even for certification, the *RSALib* is provided only as object files, without the source code [BSI15].

The rest of the chapter is organized as follows: Section 4.2 is intended for readers with interest in the mathematical details of the discovered flaw and the proposed factorization and fingerprinting methods. The readers interested mainly in the practical impacts should focus on the specifics of the implementation of the factorization method covered in Section 4.3 and the survey of impacted usage domains and the analysis of vulnerable keys found in the wild, as provided in Section 4.4. The possible approaches to short- and long-term mitigation are discussed in Section 4.5. Related work and conclusions are provided in Sections 4.6 and 4.7, respectively.

4.2 Fingerprinting and factorization

A factorization attack attempts to obtain the private primes p and q from the knowledge of the public modulus N. Such an attack is believed to be computationally infeasible for sufficiently long N. The factorization can be sped up if some additional information about the private exponent d or about the primes p or q is known by the attacker.

4.2.1 Format of the constructed primes

Our motivation for a deeper analysis of the keys produced by *Manufacturer*'s devices stemmed from the observation of interesting statistical properties extracted from a large number of keys as described in [Šve+16a]. When compared to other implementations and theoretical expectations on distribution of prime numbers, the keys exhibited a non-uniform distribution of $(p \bmod x)$ and $(N \bmod x)$ for small primes x. In this work, we recovered the structure responsible for the properties. All RSA primes (as well as the moduli) generated by the *RSALib* have the following form:

$$p = k * M + (65537^a \bmod M). \tag{4.1}$$

The integers k, a are unknown, and RSA primes differ only in their values of a and k for keys of the same size. The integer M is known and equal to some primorial $M = P_n\#$ (the product of the first n successive primes $P_n\# = \prod_{i=1}^{n} P_i = 2 * 3 * \cdots * P_n$). The value of M is related to the key size, where the key size is a multiple of 32 bits for keys generated by the *RSALib*. The value $n = 39$ (i.e., $M = 2 * 3 * \cdots * 167$) is used to generate primes for an RSA key with a key size within the $[512, 960]$ interval. The values $n = 71, 126, 225$ are used for key sizes within intervals $[992, 1952]$, $[1984, 3936]$, $[3968, 4096]$.

The most important property of the keys is that the size of M is large and almost comparable to the size of the prime p (e.g., M has 219 bits for the 256-bit prime p used for 512-bit RSA keys). Since M is large, the sizes of k and a are small (e.g., k has $256 - 219 = 37$ bits and a has 62 bits for 512-bit RSA). Hence, the resulting RSA primes suffer from a significant loss of entropy (e.g., a prime used in 512-bit RSA has only 99 bits of entropy), and the pool from which primes are randomly generated is reduced (e.g., from 2^{256} to 2^{99} for 512-bit RSA).

The specific format of the primes has two main consequences:

1. **Fingerprinting:** The keys are fingerprinted based on the existence of a discrete logarithm $\log_{65537} N \bmod M$. While the size of M is large, the logarithm can be computed easily since M has small factors only. The keys generated by the *RSALib* can be identified with a negligible error and within microseconds.

2. **Factorization:** During the factorization of N, we are looking for values of a, k. A naïve approach would iterate through different values of a (treating the value of $65537^a \bmod M$ as the "known bits") and apply Coppersmith's algorithm to find the unknown k, but the number of attempts is infeasibly large, as shown in Table 4.1. We found an alternative representation of the primes in question using smaller M' values (divisors of M), leading to a feasible number of guesses of the value a'. The reduction of M is possible since the entropy loss is sufficiently high to have more than enough known bits for the application of Coppersmith's algorithm to lengths including 1024 and 2048 bits.

4.2.2 Fingerprinting

The public RSA modulus N is a product of two primes p, q. The *RSALib* generates primes of the described form (4.1). The moduli have the corresponding form:

$$N = \overbrace{(k * M + 65537^a \bmod M)}^{p} * \overbrace{(l * M + 65537^b \bmod M)}^{q}, \quad (4.2)$$

for $a, b, k, l \in \mathbb{Z}$. The previous identity implies

$$N \equiv 65537^{a+b} \equiv 65537^c \pmod{M}, \quad (4.3)$$

for some integer c. The public modulus N is generated by 65537 in the multiplicative group $\mathbb{Z}_M^*$. The existence of the discrete logarithm $c = \log_{65537} N \bmod M$ is used as the fingerprint of the public modulus N generated by the *RSALib*.

Efficiency

Although the discrete logarithm problem is a hard problem in general, in our case, it can be computed within microseconds using the Pohlig-Hellman algorithm [PH06]. The algorithm can be used to efficiently compute a discrete logarithm for a group G, whose size $|G|$ is a smooth number (having only small factors). This is exactly our case with the group $G = [65537]$ (subgroup of $\mathbb{Z}_M^*$ generated by 65537). The size of G is a smooth number (e.g., $|G| = 2^4 * 3^4 * 5^2 * 7 * 11 * 13 * 17 *$

$23 * 29 * 37 * 41 * 53 * 83$ for 512-bit RSA) regardless of the key size. The smoothness of G is a direct consequence of the smoothness of M. Since M is smooth (M is a primorial, $M = 2 * 3 * 5 * \cdots * P_n$), the size of $\mathbb{Z}_M^*$ is even "smoother" ($|\mathbb{Z}_M^*| = \varphi(M)$). The size $|G|$ is a divisor of $|\mathbb{Z}_M^*|$ (from Lagrange's theorem), and it is therefore smooth as well.

False positives

The existence of the discrete logarithm serves as a very strong fingerprint of the keys. The reason is that while random primes/moduli modulo M cover the entire $\mathbb{Z}_M^*$, the *RSALib* generates primes/moduli from the group G – a tiny portion of the whole group. The sizes $|G|$ of the group G are listed in Table 4.1 in the Naïve brute force (BF) column. The size of $\mathbb{Z}_M^*$ is equal to $\varphi(M)$. For example, $|G| = 2^{62.09}$ while $|\mathbb{Z}_M^*| = \varphi(M) = 2^{215.98}$ for 512-bit RSA. The probability that a random 512-bit modulus N is an element of G is $2^{62-216} = 2^{-154}$. This probability is even smaller for larger keys. Hence, we can make the following conclusion with high confidence: an RSA key was generated by the *RSALib* if and only if the Pohlig-Hellman algorithm can find the discrete logarithm $\log_{65537} N \bmod M$. Our theoretical expectation was verified in practice (see Section 4.3.1) with no false positives found within a million of tested keys.

4.2.3 Factorization – attack principle

Our method is based on Coppersmith's algorithm, which was originally proposed to find small roots of univariate modular equations. In [Cop96a], Coppersmith showed how to use the algorithm to factorize RSA modulus N when high bits of a prime factor p (or q) are known. We slightly modified the method to perform the factorization with known $p \bmod M$ ($= 65537^a \bmod M$).

Coppersmith's algorithm

Coppersmith's algorithm is used as a parametrized black box in our approach. Parameters affect the success rate and running time of the algorithm. In order to optimize the entire factorization process, we optimized the parameters of Coppersmith's algorithm. We choose

parameters so that the algorithm will certainly find unknown bits of the factor and so the computation time will be minimal. The fraction of known bits of the factor determines the optimal parameters (100% success rate, best speed) of the algorithm. Coppersmith's algorithm is slowest when using the required minimum of known bits (half of the bits of the factor). With more bits known, the running time of the algorithm decreases.

Naïve algorithm

For N of the form (4.2), we look for factor p or q. In order to find a prime factor (say, p), one has to find the integers k, a. A naïve algorithm would iterate over different options of $65537^a \mod M$ and use Coppersmith's algorithm to attempt to find k. The prime p (q, respectively) is found for the correct guess of parameter a (b). The cost of the method is given by the number of guesses (*ord*) of a and the complexity of Coppersmith's algorithm. The term *ord* represents the multiplicative order of 65537 in the group $\mathbb{Z}_M^*$ ($ord = ord_M(65537)$) and can be computed simply using the technique described in Section 4.2.6. In practice, *ord* determines the running time of the entire factorization. The number of attempts is too high (see Table 4.1, *Naïve BF # attempts*) even for small key sizes. Decreasing the number of attempts is necessary to make the method practical.

Main idea

A crucial observation for further optimization is that the bit size of M is analogous to the number of known bits in Coppersmith's algorithm. It is sufficient to have just $log_2(N)/4$ bits of p for Coppersmith's algorithm [Cop96a]. In our case, the size of M is much larger than required ($log_2(M) > log_2(N)/4$). The main idea is to find a smaller M' with a smaller corresponding number of attempts $ord_{M'}(65537)$ such that the primes are still of the form (4.1), with M replaced by M' and a, k replaced by a', k'. The form of the primes p, q implies that the modulus N is of the form (4.2) and (4.3) also for M' – of course with new corresponding variables a', b', c', k', l'.

In order to optimize the naïve method, we are looking for M' such that:

1. primes (p, q) are still of the form (4.1) – M' must be a divisor of M;

2. Coppersmith's algorithm will find k' for correct guess of a' – enough bits must be known ($log_2(M') > log_2(N)/4$);

3. overall time of the factorization will be minimal – number of attempts ($ord_{M'}(65537)$) and time per attempt (running time of Coppersmith's algorithm) should result in a minimal time.

For practical factorization, there is a trade-off between the number of attempts and the computational time per attempt as Coppersmith's algorithm runs faster when more bits are known (see Figure 4.2). In fact, we are looking for an optimal combination of value M' and parameters (m, t – for more details, see Section 4.2.7) of Coppersmith's algorithm. It should be noted that the search for value of M' is needed only once for each key size. The optimal parameters M', m, t along with N serve as inputs to our factorization Algorithm 1.

Input : N, M', m, t
Output: p – factor of N
$c' \leftarrow \log_{65537} N \bmod M'$ ▷ Use Pohlig–Hellman alg;
$ord' \leftarrow ord_{M'}(65537)$ ▷ See Section 4.2.6 for method;
forall $a' \in \left[\frac{c'}{2}, \frac{c'+ord'}{2}\right]$ **do**
  $f(x) \leftarrow x + (M'^{-1} \bmod N) * (65537^{a'} \bmod M') \pmod N$;
  $(\beta, X) \leftarrow (0.5, 2 * N^{\beta}/M')$ ▷ Setting parameters;
  $k' \leftarrow Coppersmith(f(x), N, \beta, m, t, X)$;
  $p \leftarrow k' * M' + (65537^{a'} \bmod M')$ ▷ Candidate for a factor;
  if $N \bmod p = 0$ **then**
    return p
  end
end

Algorithm 1: The factorization algorithm for RSA public keys N generated by the *RSALib*. The input of the algorithm is a modulus N of the form (4.1) with M' as a product of small primes and optimized parameters m, t for Coppersmith's method.

Results

The optimized values of M' for different key lengths were found along with parameters m, t using a local brute force search optimized by the results of a greedy heuristic. The size of the resulting M' is more than the bound $log_2(N)/4$ but is relatively close to it. The resulting order (Table 4.1, *Optimized BF # attempts*) is small enough for the factorization of 512, 1024 and 2048-bit RSA to be practically feasible. Figure 4.1 summarizes the factorization complexity and relevant parameters for all key lengths between 512 and 4096 bits with 32-bit steps. The search space of a' can be trivially partitioned and parallelized on multiple CPUs. We verified the actual performance of the proposed factorization method on multiple randomly selected public keys.

Key size	512 b	1024 b	2048 b	3072 b	4096 b
M	$P_{39}\# = 167\#$	$P_{71}\# = 353\#$	$P_{126}\# = 701\#$	$P_{126}\# = 701\#$	$P_{225}\# = 1427\#$
Size of M	219.19 b	474.92 b	970.96 b	970.96 b	1962.19 b
Size of M'	140.77 b	285.19 b	552.50 b	783.62 b	1098.42 b
Naïve attempts	$2^{61.09}$	$2^{133.73}$	$2^{254.78}$	$2^{254.78}$	$2^{433.69}$
Our attempts	$2^{19.20}$	$2^{29.04}$	$2^{34.29}$	$2^{99.29}$	$2^{55.05}$
Time/attempt	11.6 ms	15.2 ms	212 ms	1159 sec	1086 ms
Worst case	1.9 CPU h	97 CPU days	141 years	$2.8 * 10^{25}$ years	$1.3 * 10^{9}$ years

Table 4.1: Overview of the used parameters (original M and optimized M') and performance of our factorization algorithms for commonly used key lengths. Time measurements for multiple attempts were taken on one core of an Intel Xeon E5-2650 v3 CPU clocked at 3.00 GHz, and the worst case time estimates are extrapolated from the orders and the average times required per attempt. The expected factorization time is half of the worst case time.

4.2.4 Coppersmith's algorithm in detail

There are various attacks on RSA based on Coppersmith's algorithm (for a nice overview, see [May09]). The algorithm is typically used in scenarios where we know partial information about the private key (or message) and we want to compute the rest. The given problem is

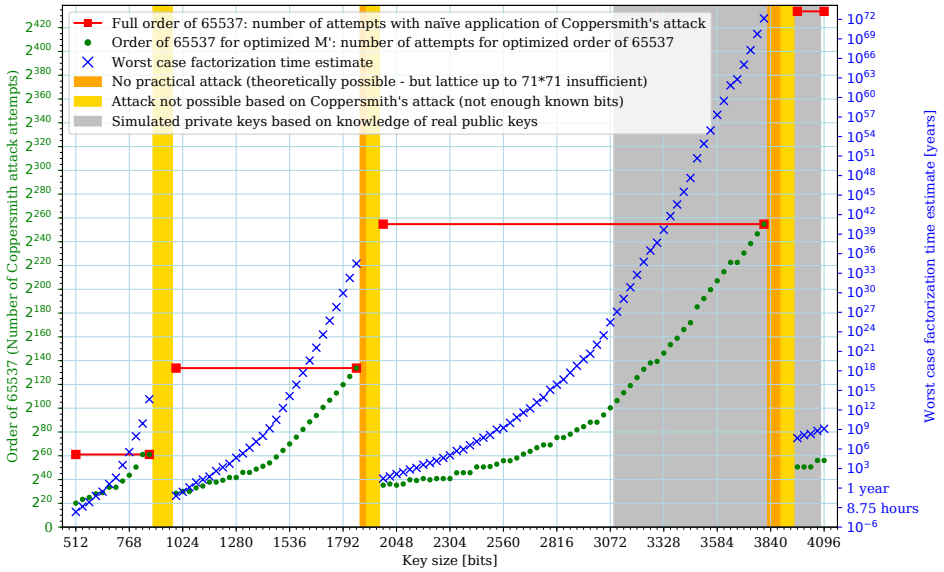

Figure 4.1: The complexity of the factorization of keys produced by the studied *RSALib* with different key lengths starting from 512 to 4096 bits in 32-bit steps (horizontal axis). The blue crosses show the worst case estimate for the time to factorize a key with the given length, with the vertical axis scale on the right side showing the estimated CPU effort on one core of an Intel Xeon E5-2650 v3 CPU clocked at 3.00 GHz. The red lines show the full order of the group. The green dots show the reduced order as achieved by our method. The yellow areas indicate the key lengths for which our method, which is based on Coppersmith's attack, is not applicable due to an insufficient number of known bits. The orange areas indicate the key lengths where the attack should be possible in practice; however, we were not successful in finding suitable parameters. The gray area shows the key lengths where only public keys were available to us; hence, we simulated the private keys for the computations backing the creation of the graph (since the structure of the keys can be recovered from either private or public keys, the simulation should be sufficient).

solved in the three steps:

$$problem \quad \rightarrow \quad f(x) \equiv 0 \pmod{p} \quad \rightarrow \quad g(x) = 0 \quad \rightarrow \quad x_0$$

First, we transform the given *problem* to the modular polynomial equation $f(x) \equiv 0 \pmod{p}$ with an unknown p (divisor of some known N) and the small root x_0 ($f(x_0) \equiv 0 \pmod{p}$) we are looking for. The root x_0 should be smaller than some sufficiently small constant X (i.e., $|x_0| < X$). The polynomial $f(x) \in \mathbb{Z}[x]$ should be constructed so that the root x_0 solves the *problem*. In the second step, Coppersmith's algorithm eliminates the unknown p by transforming the modular equation to the equation $g(x) = 0$ over the integers that have the same roots (i.e., x_0 is a root of $g(x)$). In the third step, all roots x_0 of the integer polynomial $g(x)$ are found easily by standard methods (e.g., the Berlekamp-Zassenhaus algorithm [Ber67; CZ81]).

The polynomial $g(x)$ is constructed in Coppersmith's algorithm as a linear combination $g(x) = \sum_i a_i * f_i(x), a_i \in \mathbb{Z}$ of some polynomials $f_i(x)$ derived from $f(x)$. The polynomials f_i are chosen so that $f_i(x)$ and $f(x)$ have the same roots modulo p. This implies that $g(x)$ and $f(x)$ share the same roots (with some additional roots) modulo p as well. The main idea of Coppersmith's algorithm is to find $g(x) \in \mathbb{Z}[x]$ such that $|g(x_0)| < p$, which means that the equivalence $g(x_0) \equiv 0 \pmod{p}$ also holds over the integers, i.e., $g(x_0) = 0$. The polynomial $g(x)$ is found by the LLL algorithm [LLL82] using the fact that the root x_0 is small.

The LLL algorithm reduces a lattice basis $b_0, \cdots, b_{n-1}$. The algorithm computes an alternative basis $b'_0, \cdots, b'_{n-1}$ of the lattice such that the vectors $b'_0, \cdots, b'_{n-1}$ are smaller than the vectors in the original basis. The LLL algorithm is typically used to find one sufficiently short vector of the lattice. The algorithm is applied to the matrix B, which consists of row vectors b_i. The result of the reduction is the matrix B' of short vectors b'_j, which are all constructed as linear (but with integer coefficients) combinations of basis vectors b_i. Coppersmith's algorithm utilizes the LLL algorithm in order to find the desired polynomial $g(x)$ with a small function value $g(x_0)$. The LLL is used to find an "equivalent" polynomial $g(xX)$ (x – a variable, X – a known constant) rather than $g(x)$. The LLL is used here to find the polynomial $g(xX)$ as a linear combination of polynomials $f_i(xX)$. The LLL algorithm is applied to the matrix B that consists of coefficient vectors of polynomials $f_i(xX)$ for $|x_0| < X$. The polynomial $g(x)$ is defined by the smallest vector b'_0 of the reduced basis as $g(x) = \sum_{i=0}^{n-1} b'_{0,i} x^i$

for $b'_0 = [b'_{0,0}, b'_{0,1}, \cdots, b'_{0,n-1}]$. A small norm $|b'_0| = \sqrt{\sum_{i=0}^{n-1}(b'_{0,i}X^i)^2}$ of the vector b'_0 implies small function value $|g(x_0)| = |\sum_{i=0}^{n-1} b'_{0,i}x_0^i|$ [May09, Proof of Theorem 2].

4.2.5 Application of Coppersmith's algorithm

Our factorization is based on the SageMath implementation [Won15] of the Howgrave-Graham method [How97] – a revisited version of Coppersmith's algorithm.

Howgrave-Graham method

In general, Coppersmith's algorithm and the Howgrave-Graham method use p^m instead of p, i.e., x_0 is root of polynomials $f_i(x)$ modulo p^m, and we are looking for $g(x)$ such that $|g(x_0)| < p^m$. The method uses the following set of polynomials $f_i(x)$ generated as:

$$f_i(x) = x^j N^{m-i} f^i(x) \quad i = 0, \cdots, m-1, \quad j = 0, \cdots, \delta - 1, \tag{4.4}$$

$$f_{i+m}(x) = x^i f^m(x) \qquad i = 0, \cdots, t-1, \tag{4.5}$$

and parametrized by the degree δ of the original polynomial $f(x)$ ($\delta = 1$ in our case). The Coppersmith-Howgrave-Graham method is further parametrized by three parameters m, t, X (apart from $f(x), N$), defining the matrix B and influencing the running time. The parameters m, t define the number of polynomials $n = \delta * m + t$ and the dimension of the square matrix B. The third parameter X – the upper bound for the solutions we are looking for ($x_0 < X$) – determines the bit size of the entries of the matrix B. The running time of Coppersmith's algorithm is dominated by the time needed for the LLL reduction. The running time of the LLL reduction is given by the matrix B (the row dimension and the size of its entries) and is mostly determined by the matrix size n.

Application to ($p \bmod M'$) known

The Howgrave-Graham method is able to find a sufficiently small solution x_0 of the equation $f(x_0) \equiv 0 \pmod{p}$. In our case, the primes p, q are of the form $p = k' * M' + (65537^{a'} \bmod M')$, with the small k'

being the only unknown variable of the equation. Hence, the polynomial $f(x)$ can be constructed as $f(x) = x * M' + (65537^{a'} \bmod M')$, since $f(x_0) = 0 \bmod p$ has a small root $x_0 = k'$. The method requires $f(x)$ to be monic (the leading coefficient is 1), but the form can be easily obtained [May09] as:

$$f(x) = x + (M'^{-1} \bmod N) * (65537^{a'} \bmod M') \pmod{N}. \quad (4.6)$$

Setting the parameters X and β

The parameter β represents the upper bound for the ratio of the bit size of the factor p and the modulus N, i.e., $p < N^{\beta}$. Since the bit size of both primes p, q is half of the bit size of N, the value β is set to 0.5. The parameter X represents the upper bound for the solution x_0 of the modular polynomial equation. In our case, X represents an upper bound for the value of k' from the equation (4.1); hence, its value can be computed as $X = 2 * N^{0.5} / M'$.

4.2.6 Computing the order of a generator in $\mathbb{Z}^*_{M'}$

The order of the generator 65537 is used in our Algorithm 1 and also for the optimization of parameters (see the next section). The multiplicative order $ord' = ord_{M'}(65537)$ is the smallest non-zero integer such that $65537^{ord'} \equiv 1 \pmod{M'}$, which is equivalent to $65537^{ord'} \equiv 1 \pmod{P_i}$ for all prime divisors P_i of M'. Since M' is the product of different primes, the ord' can be computed as the least common multiple of the partial orders $ord_{P_i} = ord_{P_i}(65537)$ for primes divisors P_i of M':

$$ord' = lcm(ord_{P_1}, ord_{P_2}, \cdots) \text{ for } P_i | M'. \quad (4.7)$$

4.2.7 Optimization of the parameters M', m, t

The optimization of parameters is performed only once for all RSA keys of a given size. The parameters M', m, t affect the success rate and the running time of our method. We are looking for parameters M', m, t such that the success rate of our empirical evaluation is 100% and the overall time is minimal. The success rate is measured by verifying that

k' is found for a correct guess of a' for each key in our testing sample. The running time (the worst case)

$$Time = ord_{M'}(65537) * T(M', m, t)$$

of our method is determined by the number of guesses ($ord_{M'}(65537)$ for the parameter a') and the computation average running time T for one attempt (computation of k' using Coppersmith's algorithm). The running time of Coppersmith's algorithm is dominated by the LLL reduction so it is affected mostly by the size $n = m + t$ of the square matrix B and partially by the size of matrix elements given by the size of M'. The value M' affects both the number of attempts ($ord_{M'}(65537)$) and the time for one attempt ($T(M', m, t)$); hence, we optimize all parameters M', m, t together. During the optimization, we focus on decreasing the value $ord' = ord_{M'}(65537)$. The optimization process can be described as follows:

1. Compute a set of candidates for M', each candidate with sufficiently small corresponding ord';

2. For each candidate, find the optimal (100% success rate, minimal time per attempt) parameters m, t – only reasonably small parameters m, t are brute-forced ($t = m + 1$ and $m = 1, \cdots, 35$). For a given m, t, the Howgrave-Graham method is applied to the polynomial (4.6) for correct guess of a' (to compute success rate) and also for incorrect guesses of a' (to compute the average time per attempt);

3. Return the best combination M', m, t with the minimal corresponding *Time*.

The best combination M', m, t was obtained with respect to the implementation [Won15] of the Howgrave-Graham method applied to keys of a given size. We used a dataset of RSA keys of given sizes (512 to 4096 bits, by 32-bit increments) with known factorizations and having our special form (4.2). The approximate size of the optimized M' for various key lengths can be found in Figure 4.1. The most common key lengths used the following m, t values: $m = 5, t = 6$ for 512, $m = 4, t = 5$ for 1024, $m = 6, t = 7$ for 2048, $m = 25, t = 26$ for 3072, $m = 7, t = 8$ for 4096.

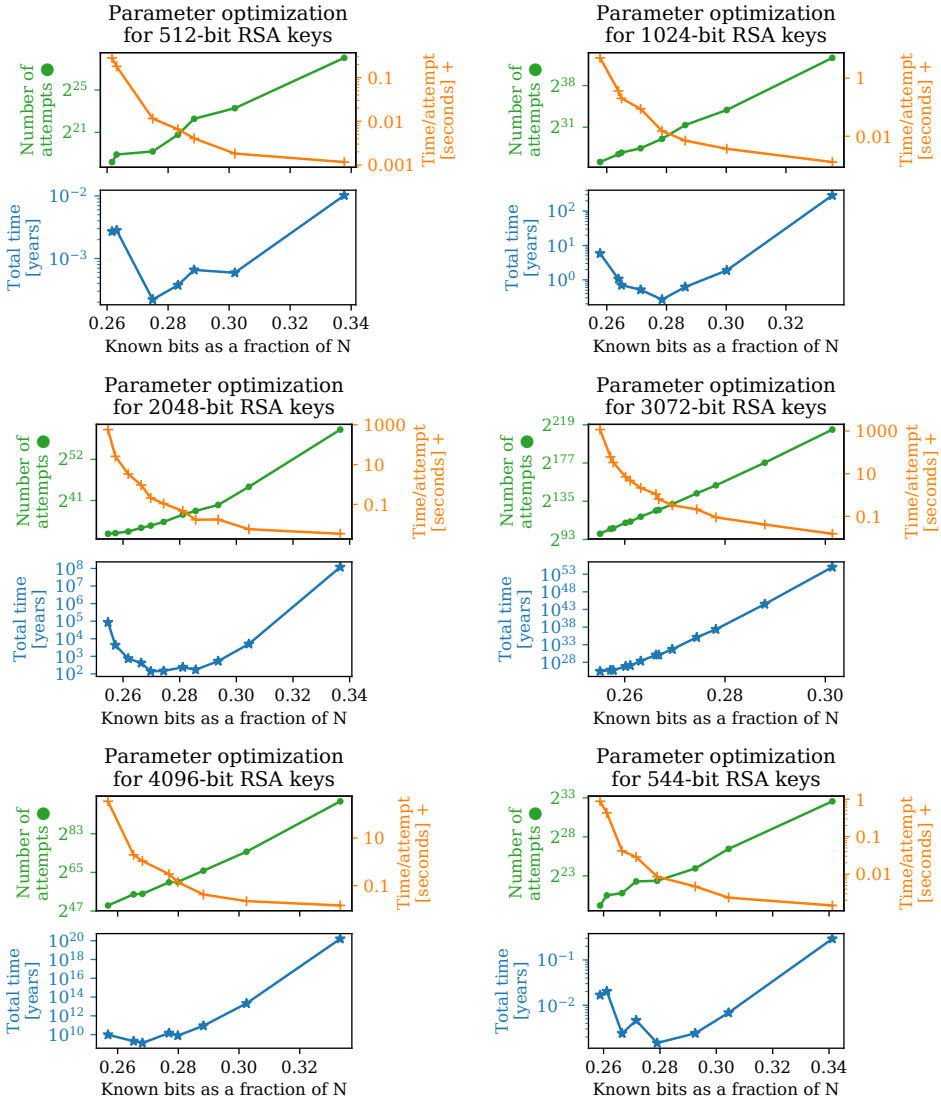

Figure 4.2: The trade-off between the number of attempts (green circles) and the time per attempt (orange crosses) as the function of the number of known bits (size of the optimized M'). We select the minimal overall time of the factorization (blue stars). Values with the same lattice size perform an attempt in an approximately same time. The best number of attempts for each considered lattice size ($m + t$) is plotted. There can be more than one local minimum for time (e.g., see 544-bit RSA keys). The vertical axis has logarithmic scale.

Input : primorial M, ord' – divisor of $ord_M(65537)$
Output: M' of maximal size with $ord_{M'}(65537)|ord'$
$M' \leftarrow M$;
forall *primes* $P_i|M$ **do**
  $ord_{P_i} \leftarrow ord_{P_i}(65537)$;
  if $ord_{P_i} \nmid ord'$ **then**
    $M' \leftarrow M'/P_i$;
  end
end
return M'

Algorithm 2: The computation of the maximal divisor M' of the primorial M with $ord_{M'}(65537)|ord'$ for a given ord' (divisor of $ord_M(65537)$).

Optimizing M'

We aim to preserve the format of the primes so that we can efficiently generate the candidate values for known bits using the formula $(65537^c \bmod M')$. To accomplish that, we are looking for a divisor M' of M (see Section 4.2.3) that is a primorial $M = 2 * 3 * \cdots * P_n$. Divisor M' of M is selected as a candidate for an optimal M' (with the best m, t) if the value $ord_{M'}(65537)$ is sufficiently small but the size M' is large enough (Coppersmith's algorithm requires $log_2(M') > log_2(N)/4$).

Our aim is to perform a brute force search for M'. In order to speed up the search we are looking for the value $ord_{M'}(65537)$ rather than M'. Once $ord' = ord_{M'}(65537)$ is found, the maximal corresponding value M' can be computed easily. Although the search space for ord' is smaller than the space for M', the brute force search is still feasible only for smaller key sizes. Hence, we used a combination of two heuristics – greedy and local brute force search.

The general strategy is to maximize the size of M' and simultaneously minimize the corresponding order. The value M' for given key size was found in two steps:

- First, we used a greedy heuristic (with a "tail brute force phase") to find an "almost" optimal M', denoted by M'_{greedy} with the corresponding order ord'_{greedy}.

- Second, the value ord'_{greedy} was used to reduce the search space of a "local" brute force search for a better M'.

In both strategies, we used the simple Algorithm 2 that given ord' looks for the maximal M' (divisor of M) such that given ord' equals $ord_{M'}(65537)$. In some cases, no such M' exists, then Algorithm 2 finds M' such that the corresponding order $ord_{M'}(65537)$ is the maximal proper divisor of given ord'. Algorithm 2 is based on the formula (4.7). The algorithm eliminates from M only those prime divisors $P_i|M$ whose partial order $ord_{P_i}(65537)$ does not divide given ord'.

Greedy heuristic

In the greedy strategy, we try to minimize $ord_{M'}(65537)$ and simultaneously maximize the size of M' (to get $log_2(M') > log_2(N)/4$). The greedy heuristic is an iterated strategy with local optimal improvement performed in each iteration. In each iteration, we reduce (divide) ord' by some prime power divisor $p_j^{e_j}$ and compute the corresponding M' of maximal size using Algorithm 2. In the greedy choice, we select the most "valuable" prime power divisor $p_j^{e_j}$ of ord' that provides a large decrease in the order ord' at a cost of a small decrease in the size of M'. The divisor is chosen as the highest reward-at-cost value, defined as:

$$\frac{\Delta\text{size of } ord_{M'}}{\Delta\text{size of } M'} = \frac{\log_2(ord_{M'_{old}}) - \log_2(ord_{M'_{new}})}{\log_2(M'_{old}) - \log_2(M'_{new})}$$

for M'_{new} computed by Algorithm 2 with M'_{old}, $ord' = ord_{M'_{old}}/p_j^{e_j}$ as an input. The reward-at-cost represents the bit size reduction of the order at the cost of the bit size reduction of M'. The following example illustrates how our greedy heuristic works:

Example 1. *The initial M'_{old} for RSA–512 is set to $M = P_{39}\# = 167\# = 2 * 3 * \cdots * 167$. The factorization of the initial order is: $ord' = ord_{M'_{old}} = 2^4 * 3^4 * 5^2 * 7 * 11 * 13 * 17 * 23 * 29 * 37 * 41 * 53 * 83$. There are 20 candidates $2^1, \cdots, 2^4, 3, \cdots, 3^4, 5, 5^2, 7, \cdots, 83$ for the most valuable prime power divisor $p_j^{e_j}$ of ord' in the first iteration. For the candidate $p_j^{e_j} =$*

83^1, *Algorithm 2 eliminates 167 from* M'_{old} *since* $83^1 | ord_{167} = 166$ *and* $83^1 \nmid ord'$. *Algorithm 2 returns* $M'_{new} = M'_{old}/167$ *for the input values* $M_{old}, ord' = ord_{M'_{old}}/83^1$. *The reward-at-cost for* 83^1 *is computed as* $\log_2 83/\log_2 167 = \frac{6.37}{7.38}$ *for the reduction of the order by 6.37 bits and the reduction of* M' *by 7.38 bits. For the candidate* 17^1, *Algorithm 2 eliminates* $103, 137$ *(i.e.,* $M'_{new} = M'_{old}/(103 * 137)$*), since* $17 | ord_{103} = 51 = 17 * 3, 17 | ord_{137} = 136 = 17 * 8$ *and* $17^1 \nmid ord_{M'_{old}}/17$. *The reward-at-cost for* 17^1 *is computed as* $\log_2 17/\log_2(103 * 137) = 4.08/13.78$, *etc. The most valuable candidate in the first iteration is* $p_j^{e_j} = 83^1$ *with the highest reward-at-cost 0.8633.*

In the second iteration, we start with $M'_{old} = M/167$ *and* $ord' = ord_{M'_{old}} = 2^4 * 3^4 * 5^2 * 7 * 11 * 13 * 17 * 23 * 29 * 37 * 41 * 53$ *and compute the new reward-at-cost for all 19 candidates* $2^1, \cdots, 2^4, 3, \cdots, 3^4, 5, 5^2, 7, \cdots, 41$. *In the second iteration, the best candidate for divisor* $p_j^{e_j}$ *of new* ord' *with the highest reward-at-cost is* $p_j^{e_j} = 53^1$, *therefore in the third iteration we start with* $M'_{old} = M/167/107$, *etc.*

Throughout the iterations, the following best candidates for $p_i^{e_i}$ *are found:* $83^1, 53^1, 41^1, 29^1, 37^1, 23^1, 17^1, 3^2, 11^1$. *The value of M is divided by these respective numbers:* $167, 107, 83, 59, 149, 47 * 139, 103 * 137, 19 * 109 * 163$.

In the last iteration, the greedy heuristic computes M' *that is too small (*$\log_2(M') < \log_2(N)/4$*), which finishes the computation. The resulting* $M' =$ `0x1b3e6c9433a7735fa5fc479ffe4027e13bea` *from the previous iteration is computed by Algorithm 2 for* $M' = M/167/107/\cdots/(19 * 109 * 163)$.

The greedy method returns an optimal M' for 512-bit RSA keys. The optimality was verified by brute force, testing all possible divisors of M with sufficiently small corresponding order.

Tail brute force

The greedy strategy can be improved for larger keys by brute force testing all divisors of $ord_{M'}$ that is found by the greedy heuristic. First, we execute the greedy strategy, that gives us the sequence of the values of $M'_0 > M'_1 > \cdots > M'_L$ from the iterations $0, 1, \cdots, L$. Then, we

use brute force (testing all divisors) for $ord_{M'_i}$, starting with M'_{L-1} and continuing with $M'_{L-2}, \cdots$, limited by reasonable running time.

Local brute force

There are two ways to perform the brute force search for an optimized M' (divisor of M). We can search through divisors M' of M, or we can use an alternative search through all divisors ord' of $ord_M(65537)$ ($M'|M \implies ord_{M'}|ord_M$) and compute the corresponding M' from ord' using Algorithm 2. We use the second approach since the search space for ord' is significantly smaller than that for M'. For example, for 512-bit RSA keys, $M = P_{167}\#$ is product of 39 primes, i.e., there are 2^{39} different divisors of M, while there are only $5^2 * 3 * 2^9 \approx 2^{15}$ different divisors of $ord_M(65537) = 2^4 * 3^4 * 5^2 * 7 * 11 * 13 * 17 * 23 * 29 * 37 * 41 * 53 * 83$.

For smaller key sizes, it is possible to search through all divisors of the order, but for large key sizes (e.g., 4096-bit RSA), the brute force strategy is infeasible and needs to be optimized. We implemented an algorithm that recursively iterates through all divisors ord' of $ord_M(65537)$. Recursion allows us to optimize the search and to skip inappropriate candidates (small M', big $ord_{M'}(65537)$) for an optimal M'.

We use two approaches that recursively iterate through orders:

- Decreasingly – In this approach we start with the full order $ord' = ord_M(65537)$, and in each iteration, we divide $ord' = ord'/p_j$ by a prime divisor of current ord'. The branch of the recursion is stopped when M' is too small ($log_2(M') < log_2(N)/4$). This approach is suitable for key sizes with bit sizes of M' close to the lower bound $log_2(N)/4$ because only several primes p_j can be eliminated from ord' and most inappropriate candidates are skipped due to a small size of M'.

- Increasingly – We start with $ord' = 1$ and in each step multiply the order $ord' = ord' * p_j$ by some prime divisor p_j of $ord_M(65537)$. When ord' is too large, we stop the given branch of the recursion and skip the worst candidates. As an upper bound for ord', we use the value $ord_{greedy} * 2^5$. This approach is suitable for key sizes for which the bit size of M' is significantly bigger

than $log_2(N)/4$ since most candidates are skipped due to the large value of ord'.

4.2.8 Guessing strategy

Our method can find the prime factor p for the correct guess x of a'. A simple incremental search $x = 0, 1, \cdots$ for a' would iterate through $ord_{M'}(65537)$ for different values of x in the worst case since

$$p \equiv 65537^{a'} \pmod{M'}.$$

Denoting $ord' = ord_{M'}(65537)$, we are looking for $x \equiv a' \pmod{ord'}$.

Since both p, q are of the same form, our method can also find the factor q for $x \equiv b' \pmod{ord'}$. Hence, our method is looking simultaneously for p and q. This fact can be used to halve the time needed to find one of the factors p, q of N. In order to optimize the guessing strategy, we are looking for the smallest subset (interval) of $\mathbb{Z}_{ord'}$ that contains either a' or b'. We use the value c' obtained during the fingerprinting (a discrete logarithm of N) to describe the desired interval. The interval is of the following form:

$$I = \left[\frac{c'}{2}, \frac{c' + ord'}{2}\right].$$

It is easy to see that either a' or b' ($c' \equiv a' + b' \pmod{ord'}$) occur in the interval I and that the size of the I is the smallest possible.

4.3 Practical implementation

We implemented the full attack in SageMath, based on an implementation [Won15] of the Howgrave-Graham method [How97]. We used it to verify the applicability of the method on real keys generated on the vulnerable smartcards. It was also used to perform time measurements in order to optimize our parameters and evaluate the worst case running time, as captured by Figure 4.1 and Table 4.1.

4.3.1 Details and empirical evaluation

The fingerprint verification algorithm computes the discrete logarithm of a public modulus. We chose the primorial of 512-bit RSA as the

Key size	University cluster	Rented Amazon c4 instance	Energy-only price ($0.2/kWh)
512 b	1.93 CPU hours ✓	0.63 hours, $0.063	$0.002
1024 b	97.1 CPU days ✓	31.71 days, $76	$1.78
2048 b	140.8 CPU years	45.98 years, $40,305	$944
3072 b	$2.84 * 10^{25}$ years	$9.3 * 10^{24}$ years, $\$8.1 * 10^{27}$	$\$1.90 * 10^{26}$
4096 b	$1.28 * 10^{9}$ years	$4.2 * 10^{8}$ years, $\$3.7 * 10^{11}$	$\$8.58 * 10^{9}$

Table 4.2: An estimation of factorization times and prices for different key lengths on different types of computational devices. All results are the worst case estimates with expected resources spent being the half of the values shown. The time values marked with a check-mark (✓) were practically verified by factorization of real test keys while others were extrapolated based on a know number of attempts and a time per attempt. The energy consumption was estimated based on the thermal design power (TDP) specifications of Intel Xeon E5-2660 v3 @ 2.60 GHz [Int14] (note that peak power can be up to 1.5-3x more), time per attempt as benchmarked on Amazon c4 instance, energy price of $0.2/kWh and scaled to 2.90 GHz (as Amazon c4 uses publicly unreleased Intel Xeon E5-2666 v3 clocked at a slightly higher frequency). The university cluster column captures the factorization times as measured by us on a university computational cluster with Intel Xeon E5-2650 v3 @ 3.00 GHz CPUs scaled to a single-core of this CPU. The Amazon c4 instance price corresponds to outsourcing of a single key factorization to Amazon AWS (c4 price is $0.1/hour for a 2-core CPU). We performed benchmark on a c4 instance for a single Coppersmith's computation and extrapolated to number of attempts in the worst case. The energy-only price corresponds to situation when one operates own hardware and wants to factorize so many keys that the price of hardware completely amortizes over all factorized keys. A factorization benchmark on Microsoft Azure was also performed with results roughly comparable to Amazon AWS (+10%).

modulus, since it applies to all key lengths. We recorded no false negatives in 3 million vulnerable keys generated by *RSALib*, since all of the keys have the sought structure. As expected, no false positives

were recorded on 1 million non-affected keys generated by OpenSSL. We estimated the probability of a false positive on a single key as 2^{-154} in Section 4.2.2.

We practically verified the factorization method on multiple randomly selected 512 and 1024-bit keys. Since the complexity of factorization of a 2048-bit key could be approximately 100 CPU years, we did not select keys randomly. Instead, we generated keys on an affected smartcard and exported the private keys. The knowledge of the primes allows us to precisely compute the number of attempts required for the factorization as the distance of the initial guess $c'/2$ (Section 4.2.8) to a' or b' (whichever is closer). Out of 137,000 freshly generated keys, we selected 24 public keys with the least effort required (all keys with 2^{21} attempts or fewer) for factorization and ran the computation, each finishing within one week. We used the time measurements to verify the linear relationship of factorization time on the order and we checked that the worst case time estimate matches the slope of the line.

4.3.2 Possible improvements and limitations

The attack can be trivially parallelized on multiple computers. Each individual task is assigned a different subrange of the values a' that need to be guessed. The expected wall time of the attack can be decreased linearly with the number of CPUs (assuming that each task can execute the same number of attempts per a unit of time). However, the expected CPU time and the worst case CPU time remain unaffected.

The time of each attempt is dominated by lattice reduction. Our implementation uses the default implementation of LLL in SageMath (backed by the fpyLLL wrapper for fpLLL [The16]). A more efficient implementation might speed up the process. However, we do not expect significant improvements.

In our opinion, the best improvement could be achieved by a better choice of polynomials in the phase of lattice construction. We follow the general advice for polynomial choice from [May09]. More suitable lattice may exist for our specific problem.

Our algorithm for optimizing the running time utilizes a heuristic for finding an optimized value of the modulus M'. A better heuristic

or a bruteforce search might find a modulus, where the generator has a lower order or could discover a better combination of the lattice size and M' value.

Despite an extensive search for better values within a significantly larger space (Section 4.2.7), we obtained only small improvements of the overall factorization time (halving the overall time at best in comparison to the greedy algorithm). We examined the trade-off between the number of attempts and the time per attempt, as captured by Figure 4.2 to understand the nature of the optimization process.

We did not explore implementations of lattice reduction backed by dedicated hardware or GPUs. Most key lengths are processed with a lattice of low dimensions, however, some improvements may be gained for lengths that require a large lattice [Her+10]. In our experience, the memory used by one factorization was up to 300 MB. SageMath is an interpreted language, so the requirements of a hardware circuit might be different.

Finally, we cannot rule out that a fundamentally improved approach, which would utilize the properties of keys more efficiently, will be devised.

4.4 Analysis of impacts

The discussion of impacts is far from straightforward. First, the prevalence of factorizable keys in a given usage domain is between very easy to very difficult to obtain. For example, the prevalence of fingerprinted keys used for TLS is easy to enumerate thanks to Internet-wide scans like Censys [Dur+15a]. Obtaining large datasets of public keys for usage domains for devices expected to be more vulnerable (e.g., electronic passports) is usually significantly harder given the nature of secure hardware use.

Secondly, the actual damage caused by a factorized key varies significantly between and also within the usage domains. Finally, not all key lengths are actually factorizable, and the factorization time varies significantly – hence, the security of a particular key length depends on the target domain.

We discuss the overall impact based on the following aspects:

1. Accessibility of public keys – how difficult it is for an attacker to obtain the target public key(s) for subsequent factorization attempts;

2. Total number of factorizable keys found or assumed – as detected by scans of a given usage domain;

3. Cost to factorize the keys with the lengths actually used in the target domain (as estimated in Table 4.2);

4. Implications of a successful factorization – what damage the attacker can cause.

Note that due to the varying parameter M used by the *RSALib* when generating the keys of different lengths, the difficulty of key factorization does not strictly increase with the key length (see Figure 4.1). Some shorter keys may be actually more difficult to factorize using our method than other longer keys. As an example, a 1280-bit key is more difficult to factorize than a 2048-bit key in our setting. It is crucial to survey the precise key lengths as used within the inspected domains. We take advantage of the possibility to quickly detect the key fingerprint, with quick summary of the affected domains in Section 4.4.1 and in Table 4.3 and Table 4.4 followed with additional details for every domain thereafter.

4.4.1 Summary of results

The electronic identity documents (eIDs) domain is significantly affected. Despite the general difficulty of obtaining relevant datasets with public keys from passports or eIDs that limited our analysis to only four countries, we detected two countries issuing documents with vulnerable keys. The public lookup service of *Estonia* allowed for a random sampling of the public keys of citizens and revealed that more than half of the eIDs of regular citizens are vulnerable and that all keys for e-residents are vulnerable.

The use of two-factor authentication tokens and commit signing is on the rise, yet these approaches are still adopted only by a minority of developers – but usually for the more significant projects. The analysis of the authentication keys of all GitHub developers found several

hundreds of vulnerable keys. The developers with vulnerable keys have access to crucial open-source repositories with more than 50,000 stars. Increased scrutiny should be applied to new commits before the affected users replace vulnerable keys.

Trusted Platform Modules (TPMs) provide secure hardware anchor for trusted boot. Although it is difficult to directly extrapolate the overall prevalence of chips with vulnerable keypair generation from our limited sample of 41 laptops with different TPM chips, approximately 24% were producing vulnerable keys, indicating that the domain is significantly affected. As the replacement of a chip alone is very impractical or almost impossible, organizations have to replace the whole laptop, slowing down the recovery from the problem. Importantly, TPM is used not only to facilitate trusted boot, but also to store sensitive secrets like ones necessary to access the Volume Master Key (VMK) for Microsoft BitLocker full disk encryption software [Mic13]. The possibility to factorize TPM's 2048-bit key for "sealed storage" might lead to a recovery of BitLocker's disk decryption key in the configuration using a TPM and a PIN.

The Pretty Good Privacy (PGP) keys used for digital signatures and email encryption are easy to download from PGP keyservers. We detected almost three thousand fingerprinted keys with slightly less than one thousand practically factorizable. The *Yubikey 4* token seems to be the origin for the majority of these keys as hundreds even contain identifying strings in the keyholder information and the date of generation correlates with the release date of this token.

We found only a negligible fraction of vulnerable keys in the TLS/HTTPS domain. However, all 15 unique keys found were tied to different pages with SCADA-related topics, which may point to a single provider of a SCADA remote connection platform.

We did not collect relevant datasets of public keys for authentication tokens implementing PIV standard but found at least one instance of a widely used token utilizing chips with the affected *RSALib*. Similarly, other devices (e.g., e-health and EMV payment cards) might be impacted by the described vulnerability, although we were not able to verify the impact in such domains.

We encourage the use of our tool for detecting vulnerable keys described in Section 4.5 and the notification of affected parties if found.

4.4.2 Electronic identity documents

Various citizen identity documents represent a large area for the application of cryptographic smartcards, such as biometric passports (ePassport, ICAO Doc 9303), eDriver licenses (ISO/IEC 18013) and additional identity documents. Some national IDs are based on the same suite of protocols as ePassports, which are standardized by ICAO 9303 [ICA06]. Other countries have implemented their own suite of protocols, such as the Estonian EstEID [Trü17], the Belgian eID [AZ04] or the Taiwanese ID.

Electronic passports and identity cards utilize digital signatures for: 1) the authentication of stored data (passive authentication); 2) the verification of the genuine origin of the chip inside (active authentication, AA); and 3) the establishment of a secure channel between the passport and the border inspection terminal with mutual authentication (Extended Access Control, EAC-PACE). Additionally, in some instances, the issuing country uses the national IDs for citizen authentication when accessing government services via the Internet.

The suppliers of ePassport implementations typically provide the platform in several possible configurations with different supported algorithms (RSA-based, EC-based) and protocols (EAC-PACE, AA), leaving the choice of the preferred configuration to the issuing country. The use of the *RSALib* is referenced in multiple certification documents of electronic passports of several countries.

We are not aware of any country disclosing publicly the full database of their citizens public keys. A small fraction of countries provide lookup services with significant limitations on the number of queries allowed. We analyzed four different types of digital certificates issued by the country of *Estonia*: a) regular citizenship eID keys (denoted as *esteid*); b) eID keys for electronic use only ("digital certificate of identity", denoted as esteid-*digi*); c) keys for operations from mobile devices (denoted as esteid-*mobiil*); and d) e-resident keys (denoted as esteid-*resident*). For every type, separate authentication (*auth*) and signature (*sign*) 2048-bit RSA keys are available. The keys are used to support various eGovernment services, including VAT forms, private companies management (all types) and voting (*esteid*). In total, we analyzed the keys of approximately 10% of randomly selected citizens. The results showed a mix of on-card and out-of-card key

generation. More than half of the analyzed keys were vulnerable for *esteid* and all keys were vulnerable for esteid-*digi* and esteid-*resident*. No vulnerable keys were detected for esteid-*mobiil*. Extrapolation to the whole population results in at least hundreds of thousands of vulnerable keys.

Additionally, we analyzed keys from a limited sample of keys extracted from the physical electronic documents of three other countries and detected one (*Slovakia*) issuing documents with fingerprinted 2048-bit keys.

These results also demonstrate the general difficulty of analyzing the impacted domains – large-scale analysis was possible only for the *Estonian* eID because of the public directory with more than half of the documents found to be vulnerable. The small samples collected for other countries (like *Slovakia*) give only very limited insight – are all other documents vulnerable or only a limited production series given the two vulnerable IDs detected? Or were only documents from non-vulnerable series for other countries inspected?

The possibility of factorizing on-card keys would lead to cloning of legitimate passports or identity cards. The Slovak national ID in question is also deployed in the wider context of an eGovernment system, where the on-chip generated digital signatures serve as a replacement for traditional hand-written signatures.

4.4.3 Code signing

The digital signing of applications, modules, OS distributions or code is now common. In some cases, application signing is mandatory and enforced by the platform (e.g., Android, iOS, OS drivers) or voluntarily adopted by the developers. GPG signatures can be also used to authenticate commits or tags submitted by developers to a source control system (e.g., GitHub).

GitHub

To access the Git repositories hosted on GitHub, developers can use SSH authentication as an alternative to a password for both read and write permissions. Users may also upload GPG keys for commit signing. The public keys of all users are accessible via the public

GitHub API. We analyzed the profiles of almost 25 million GitHub users and found 4.7 M SSH keys in a scan performed in February 2017.

Hundreds of fingerprinted keys were found, including keys with access to very popular repositories with up to 2,000 stars (users bookmarking the project) for user-owned repositories and more than 50,000 stars for organization-owned repositories, including repositories that are very influential in the Internet community. The impact is increased by the fact that some relevant repositories are libraries used in other projects and are essentially trusted by third-party developers.

In total, we found 447 fingerprinted keys. More than half (237) have practically factorizable key length of 2048 bits, with the rest mostly being 4096-bit RSA keys. However, it is not straightforward to determine whether a particular account has write access to repositories not explicitly owned by the account. Similarly, membership in an organization does not guarantee write access to particular repositories. GitHub does not provide this kind of information directly, and the APIs that can be used to derive this information are quite limited. The information can be inferred from an analysis of previously performed commits by the given user. We verified several instances manually and confirmed access with factorizable keys.

We view the overall impact as significant. Luckily, any potential changes made to a repository can be traced back to a particular commit due to the nature of source control systems. Many projects also use commit reviews (e.g., using pull requests), where increased caution should be used until the affected users move to more secure keys.

Maven

The Maven public repository has required developers to sign uploaded artifacts since approximately 2009 [Mak12]. Each developer must be associated with a PGP key that is also publicly reachable from a PGP keyserver. Each artifact is uniquely identified by a tuple (group ID, artifact ID, version). We downloaded the most recent versions of each artifact found in the Maven repository index in April 2017. In total, we downloaded 180,730 artifacts equipped with the Maven index file (pom.xml) – 161,841 had a signature on the pom.xml file. There were 16,959 unique PGP keys found, of which 5 were fingerprinted, all with 4096-bit moduli (not considered practically factorizable by our

method). The potentially affected artifacts appear as dependencies only in a few other artifacts. We therefore estimate the impact as small.

Android

We downloaded the 540 most popular Android applications and the 540 top ranking Android games according to the Google Play top charts. The content of the Android application package (APK) is signed with the developer key before being published to the Google Play system. There is no simple way for the developers to change the signing keys; hence, the applications will most likely have used the same keys since the time of the first upload. No fingerprinted keys were detected among the top 540 applications and games in a scan performed in January 2017. The analysis should be also extended to less popular applications. If any vulnerable keys are found in other already established applications, the affected developers may have complications migrating to different signing keys.

4.4.4 Trusted Platform Modules

Trusted Platform Module (TPM) is a specification created by the Trusted Computing Group [Tru06; Tru11]. TPMs are cryptographic hardware (usually in form of a chip attached to a motherboard) that provide basic cryptographic functionality. The typical use cases include: a) secure storage of a user's private keys or disk decryption keys; b) maintaining an unspoofable log of applications that were deployed on a target machine via a hash chain (Platform Configuration Registers – PCRs); and c) attestation of the state of the platform to a remote entity by an on-TPM signature of the PCRs. The TPM specification version 1.2 supports only RSA with 2048-bit keys [Tru06].

We analyzed a sample of 41 different laptop models equipped with TPM chips. Six different manufacturers were detected, with chips supplied by *Manufacturer* (acronym IFX) being the most common and found in 10 devices. TPM chips from devices produced before 2013 and with firmware versions[3] between 1.02 and 3.19 do not exhibit a fingerprint and are not factorizable by our method. All chips found in

3. The version of the firmware of the TPM chip does not directly relate to the version of the *RSALib*.

devices introduced in 2013 or later were vulnerable, including both TPM 1.2 and TPM 2.0. In our sample, the fingerprinted keys from the *RSALib* appear earliest in the firmware version 4.32 (however, we had no TPM chip with a version between 3.19 and 4.32 in our sample). All subsequent chip versions, including 5.x and 6.x, were also found to produce vulnerable keys. We hypothesize that the *RSALib* was first used with TPM firmware version 4.x.

There are two important RSA private keys stored inside a TPM – the *Endorsement key* (EK), which is permanently embedded by the chip manufacturer during its production and cannot be changed, and the long-term *Storage Root Key* (SRK), which is generated on-chip when a user claims the TPM ownership. Additionally, dedicated *Attestation Identity Keys* (AIKs) used for Remote Attestation may be generated.

The factorization of the EK compromises the root of trust for chip authentication. An attacker can generate a new keypair outside the TPM and then sign it with the factorized EK; hence, it will be trusted by the remote system (e.g., the company network).

The TPM can hold only a very limited number of private keys directly on the chip. All other private keys are generated inside the TPM but are then wrapped by the SRK and exported outside the TPM. If required, the keys are imported back, unwrapped and used. The factorization of the SRK therefore allows an attacker to decrypt all previously exported wrapped private keys, including the "sealed storage" packages with sensitive information otherwise readable only on the particular machine with the associated AIK keys used for Remote Attestation. If AIK is directly factorized or its value is compromised due to the factorization of the SRK, an attacker is able to forge an attestation report – allowing the attacker to start additional or modified malicious software without being noticed.

The "sealed storage" is also utilized by Microsoft BitLocker full disk encryption software [Mic13] to store a sensitive value required to obtain the Volume Master Key [Kor09; KK08]. BitLocker is typically setup together with TPM and an additional secret – either a PIN, a recovery key on a USB token, or both. The possibility to factorize TPM's 2048-bit SRK directly leads to a decryption of an unwrapping key necessary to decrypt the Volume Master Key, thus bypassing the need for TPM to validate the correctness of a PIN value via a dedicated PCR. As a result, an attacker can decrypt a disk from a stolen laptop

Domain name	Used length (bits)	Pub. key availability	Misuse
TLS/HTTPS	2048	easy	MITM/eavesdropping
Message security (PGP)	1024/2048	easy	message eavesdropping, forgery
Trusted boot (TPM)	2048	limited	unseal data, forged attestation
Electronic ID, ePassport	2048	limited	clone passport, document forgery
Payment cards (EMV)*	768/960/ 1024/1182	limited	clone card, fraudulent transaction
Certification authorities*	2048+	easy	forged certificates, MITM
Authentication tokens	2048+	limited	unauthorized access or operation
Software signing	2048+	easy	malicious application update
Smartcard (Java Card)	1024-4096	depends	depends on use

Table 4.3: The summary of the impact of key factorization in the different usage domains. The fingerprinted keys were found within all listed domains with exceptions marked with an asterisk (*). No fingerprinted keys were found in the very limited dataset of 13 EMV cards that we collected or for large datasets of browser-trusted root and intermediate CAs.

with a vulnerable TPM if encrypted by BitLocker in TPM+PIN mode (but not in a configuration with an additional USB token). We did not verify the attack in practice due to BitLocker's proprietary storage format and the cost of factorization of a 2048-bit SRK key.

4.4.5 PGP with cryptographic tokens

The private key as used in Pretty Good Privacy (PGP) [Gar95] is typically a very sensitive long-term secret. If compromised, an attacker can forge new signatures and decrypt all previously captured messages since PGP does not provide forward secrecy. Many users choose to use a cryptographic device that stores and performs private key operations inside a secure environment using an OpenPGP compliant application [Cal+07].

A large fraction of public keys used for PGP can be easily downloaded from PGP keyservers [Tub17]. Since the content of PGP key servers is publicly available, the vulnerable keys can be easily identified together with the associated user contact information. We analyzed the state of a PGP keyserver from mid-April 2017 that contained a total of 4.6 M master keys and 4.4 M sub-keys with 1.9 M and 1.7 M, respectively, being RSA keys. We detected 2,892 fingerprinted keys. Of these, two keys are 1024-bit and 954 keys are 2048-bit – both lengths

Domain name	Analyzed datasets	Vulnerable keys	
		Count	%
Complete/larger-scale datasets			
Certification authorities	all browser-trusted roots (173), level $\leqslant$ 3 intermediates (1,869)	0	0
ePass signing certificates	ICAO Document Signing Certificates, CSCA Master Lists	0	0
Estonian eID	sample of 130,152 randomly selected citizens	71,417	54.87
Estonian mobile eID	random sample of 30,471 citizens	0	0
Estonian e-residents	random sample of 4,414 e-residents	4,414	100
Message security (PGP)	complete PGP key server dump (9 M)	2,892	0.03
Software signing (GitHub)	SSH keys for GitHub developers (4.7 M)	447	0.01
Software signing (Maven)	signing keys for all public Maven artifacts	5	0.003
TLS/HTTPS	complete IPv4 scan, Certificate Transparency	15	<0.001
Trusted boot (TPM)	41 laptops with chips by 6 TPM makers	10 chips	24.39
Limited, custom-collected datasets			
Payment cards (EMV)	13 cards (4 EU countries), 6 by *Manufacturer*	0	0
Programmable smartcard	25 cards (JCAlgTest.org), 6 by *Manufacturer*	2 cards	8.67
Software signing (Android)	1,080 top ranking applications and games	0	0

Table 4.4: The summary of the number and fraction of vulnerable keys detected in different domains. The domains are ordered lexicographically and separated into two groups based on the representativeness of inspected datasets.

are practically factorizable. Additionally, 86 and 1846 fingerprinted (but not feasibly factorizable by our method) keys of 3072 and 4096-bit lengths, respectively, were detected. Finally, four keys with uncommon lengths of 3008 and 3104-bit were present.

The earliest creation date of a fingerprinted key as obtained from a PGP certificate is 2006, yet only for a single user – we hypothesize this finding was caused by an incorrect system clock. The subsequent observed year is 2009, again with a single user only. 2013 is the earliest year with keys from multiple users.

No key is observed originating in the year 2014, with more fingerprinted keys observed from July 2015 onwards. The date coincides with the official launch of a cryptographic token *Yubikey 4* (further denoted as *Token*). This hints that *Token* is the major source of the fingerprinted keys in the PGP dataset Out of 2,892 fingerprinted keys, 436 even contain some form of *Token*-related identification in the *User ID* string (154 being master keys with the rest being sub-keys). Of these, no key with a length shorter than 2048-bit is present, 96 keys are 2048-bit and 340 keys have a length of 4096 bits. Given that an

older version of *Token* is not producing fingerprinted keys, all these keys were likely generated by the newer version of *Token*.

The *Token* vendor recommends generating a keypair outside the token (for example, using OpenSSL) and importing it to facilitate private key recovery after a potential token failure. Interestingly, such advice seems not to have been followed by a significant number of users (the users who followed this advice are not detected by our fingerprinting method as their keys have no fingerprint).

The evidence for other devices (not produced directly by the *Manufacturer*) generating fingerprinted keys also shows that the *RSALib* is provided to external parties developing for the *Manufacturer* hardware.

We would like to stress that not all key lengths generated with *Token* are immediately practically factorizable by our method. *Token* can generate and use RSA keys up to 4096 bits long, which may be one of the appeals of the device – given the lack of other available smartcards supporting key lengths exceeding 2048 bits. Indeed, the analysis of the fingerprinted PGP keys with respect to the used length shows a strong user preference for 4096-bit keys. *Token* can also generate less common key lengths including 3936-bit RSA where our attack is not directly applicable, as seen in Figure 4.1. The majority of the *Token* users on this domain therefore should not be imminently affected by direct factorization using our attack, but we urge the generation of fresh keys – in light of potential further improvement of an attack.

4.4.6 TLS and SCADA-related keys

We used our fingerprinting method on two large datasets of public key certificates, used (mostly) to secure Internet TLS connections. One dataset originates from a periodic scan of the whole IPv4 address space between 2012 and 2017 [Dur+15a] collected from servers listening on port 443 and configured to prefer RSA signatures. The second dataset comes from the Certificate Transparency logs maintained by Google [Goo17c] (CT logs maintained by Google included in Google Chrome, date 2017-04-25). In total, we analyzed more than 100 million certificates.

Despite the relatively large number of keys, we only found 15 distinct fingerprinted keys – four were 1024 bits long and eleven with 2048 bits – used in tens of different certificates. Surprisingly, almost

all these certificates contain the string "SCADA" in the common name field (probably referring to Supervisory Control and Data Acquisition systems) or a URL leading to a website related to an industrial monitoring system, or both. As a result, we hypothesize that there is at least one provider of a remote connection platform with a focus on SCADA systems. It is not clear to us whether the interfaces are linked to real industrial systems since administrators of such systems may want to limit the access from the Internet. Hence, there might exist more systems with administration interfaces protected by vulnerable keys, but deployed on local networks.

Interestingly, all 15 keys contain the inspected fingerprint, but the majority of values of the most significant byte (MSB) of their moduli are significantly outside the range observed in the *RSALib* (the MSB of keys produced by the inspected smartcards and TPMs always falls in the interval `0x90-0xA8`). This finding suggests the existence of a different implementation of the prime construction algorithm with the same structure but a different modification of the most significant bits.

The factorization of the key of a TLS server trivially leads to numerous powerful attacks: server impersonation, active man-in-the-middle attack or passive decryption of the content of the communication when the connection establishment lacks forward secrecy. Overall, the impact on the public portion of the Internet seems to be only very marginal due to the small number of detected vulnerable TLS keys. However, the potentially significant impact for the entry points of some SCADA services should not be neglected.

4.4.7 Certification authorities

The presence of vulnerable keys belonging to certification authorities would magnify the impact due to the possibility of key certificate forgery. We therefore examined two significant usage domains.

Browser-trusted certificates. We examined the certificates of root certification authorities stored in Mozilla Firefox as browser-trusted roots (158 certificates) and in Ubuntu 16.04 (173 certificates). The intermediate authorities of level 1 (1,016 total), level 2 (832 total) and

level 3 (21 total) as extracted from TLS scans were also analyzed. No fingerprinted keys were detected as of May 2017.

ICAO signing certificates. We analyzed the collection of Document Signing Certificates (DSCs) of the ICAO ePassport database (version 2044) containing 8,496 certificates, and the collection of CSCA Master Lists (version 84) with 616 certificates. We also inspected the publicly available national certificates (e.g., Belgium, Estonia, Germany, Switzerland) [JMR17; Lau17] available as of May 2017. Fortunately, no vulnerable keys were found in either dataset, as the occurrence of such a certificate would lead to the possibility of impersonating an inspection terminal or forging electronic document data.

4.4.8 Generic Java Card platform

Smartcards using the Java Card platform [Mic06] have two principal configurations: 1) an open, fully programmable platform where the users develop and upload their own applications; and 2) Java Card-based systems closed from the point of view of cryptography (e.g., banking EMV or SIM cards). Here, we focus on the former configuration.

The prevalence of the *RSALib* in the area of programmable smartcards is notoriously difficult to estimate. Not all smartcards based on the *Manufacturer*'s hardware are vulnerable, as the vulnerability stems from the deployed cryptographic library and not from the hardware design itself. Many vendors use the bare hardware (e.g., SLE78 chip) and choose not to deploy the *RSALib* in question. In such a case, the implementation of the higher-level cryptographic functions (including RSA keypair generation) is done by the company that builds on the hardware produced by the *Manufacturer*. Although the vulnerable keys have a strong fingerprint that can be easily verified, the real problem (for impact assessment) lies in obtaining sample public keys. No representative public databases (comparable to those for TLS and PGP) are available.

Our analysis is based on smartcards from 10 different platform providers (Axalto, Feitian, G&D, Gemalto, Infineon, JavaCardOS, NXP, Oberthur, Softlock and Yubico) as recorded by the *JCAlgTest* database

[CRo17]. The chip manufacturer (*ICFabricator* property) and the manufacturing date (*ICFabricationDate*) can be obtained from the Card Production Life Cycle (CPLC) information as defined by the GlobalPlatform specification [Glo06].

Out of the 63 different cards included, 25 cards are listed with the provided CPLC information: 16x NXP (*ICFabricator* = 4790), 6x Infineon (4090), 1x Samsung (4250) and 2x unknown (2050 and 4070). Out of six cards with a *Manufacturer* chip, two produce fingerprinted keys. The *ICFabricationDate* property indicates the years of manufacture to be 2012 and 2015. Hence, our estimate of the prevalence of the vulnerability is confirmed again since it corresponds to the situation observed in TPM chips.

The full impact of the vulnerability will depend entirely on the scenario in which the cards are actually used. The large number of already fabricated and distributed smartcards may hinder the potential for a recall of the product from the market. The card operating system and the base libraries are stored in read-only memory and cannot be updated by the user to remove the vulnerability once a card is deployed. We expect to see the cards for a rather long time (several years) before all the vulnerable cards are eventually sold out, especially when dealing with low volume markets. The buyers should check the cards for the presence of fingerprinted keys and/or opt for longer key lengths if they are supported by the card hardware.

4.4.9 Other domains

The smartcards are also used in many other domains than those surveyed here in the previous sections, including authentication tokens; e-health cards to authenticate both patients and medical staff to access medical records or personal identity verification cards (FIPS 201 PIV [NIS13]); and electronic payment cards (EMV).

The chip-based payment cards used world-wide are backed by a set of protocols specified under the EMV standard [EMV11], which is currently maintained by the EMV consortium. The *RSALib* was approved for use in EMV cards by EMVCo [EMV12; EMV13], and we found several references to it in related certification reports. However, we are not aware of any public dataset of keys originating from EMV cards. We collected a tiny sample of RSA keys from 13 payment

cards issued by different banks in four European countries. Although 6 cards reported chips produced by the *Manufacturer*, none of them contained the distinctive fingerprint, meaning that the RSA key generation method implemented by the *RSALib* was not used in either one.

If used, the potential impact of factorizable keys would be particularly damaging to EMV cards due to the generally short RSA key lengths used. Short keys are often used for legacy reasons or to improve the usability of payments by the shorter time required to authorize the transaction (especially relevant for contactless payments). Out of the 13 cards inspected, we observed the following bit lengths of ICC keys: 768 (3x), 896 (4x), 960 (1x) and 1024 (5x).

We recommend analyzing the keys used in a particular scenario with the provided fingerprint detection tool and following the recommendations given in Section 4.5.

4.5 Mitigation and disclosure

We propose a mitigation of the attack impacts and report on the process of responsible disclosure to the *Manufacturer*.

4.5.1 Mitigation

Mitigation can be performed on multiple levels. Inarguably, algorithm replacement is the best long-term mitigation method. However, changing the algorithm requires updating firmware – which is usually not possible in already deployed devices like smartcards or TPMs with code stored in read-only memory. Other options are available even within the hardware device with the vulnerable version of the *RSALib* with some caveats. New keys can be still generated on the device if they are configured to use key lengths not directly affected by our method (yet still with a reduced security margin), or keypairs could be generated by another library (outside the device) and then imported to the device. If the potentially vulnerable keys remain deployed, their usage scenario can be supplemented with additional risk management.

Changes to the algorithm

The library could adopt an approach common in open-source libraries – instead of constructing candidates for the primes, they are generated randomly and their value is incremented until a prime is found. Other alternative constructions exist, such as provable or safe primes, as described in the NIST FIPS 186-4 standard [Nat13]. We noticed a certain similarity between the algorithm of the *Manufacturer* and an algorithm published by Joye and Paillier [JP06] focused on key generation on smartcards. The key difference seems to be the fact that the *RSALib* uses a constant value in the generator (65537), while in the paper, the value is always chosen randomly using a unit generation algorithm [JP06, Figure 4]. The approach in the paper [JP06] is *not* affected by the same vulnerability.

Note that due to the nature of deployment of the *RSALib*, some devices already in use cannot be updated. The *RSALib* is often stored in a read-only on-chip memory with no possibility to distribute and apply a fix after deployment. As an exception, the firmware of the TPMs of the *Manufacturer* can be updated.

Importing a secure keypair

A secure RSA keypair can be generated in another cryptographic library and then imported to the affected device. We are not aware of any vulnerability in *Manufacturer* devices as far as the use of securely generated keys is concerned. Coincidentally, the import of externally generated keypair is even recommended by Yubico vendor [Yub17], although for the purpose of private key backup.

Use of less affected key lengths

As discussed in Section 4.2, we consider 512, 1024 and 2048-bit keys to be insecure. Due to design choices made by the manufacturer, it appears that 3072-bit keys are seemingly less affected by our method than 4096-bit RSA though with a significantly reduced security margin. Our attack is inefficient or directly inapplicable when applied to some quite uncommon key lengths (such as 1952 bits or 3936 bits). Hence, we recommend limiting the choice of the key lengths to the seemingly unaffected keys if the usage of the vulnerable chips with on-chip

generated keys is absolutely unavoidable. Note however, that these keys still suffer from significant entropy loss. If a somewhat "standard" key length is required, we recommend switching to 3072-bit keys.

We also suggest caution when using the fingerprinted 4096-bit keys, even though our method is not practical for their factorization at the moment (requiring $1.28 * 10^9$ CPU-years). The strongest possible key length with respect to the general factorization methods and our attack is 3936-bit RSA. If a device supports at most 2048-bit keys, the key length of 1952 bits is the most secure option (see Figure 4.1).

Additional risk management

The use of potentially vulnerable keys (especially 2048-bit keys requiring feasible yet still significant computational power) can be amended with additional scrutiny to perform supplementary risk management. The presence of the fingerprint is an advantage in this scenario since the public keys can be quickly tested to decide when to apply additional measures by the cooperating system.

4.5.2 Future prevention and analysis

The impacts of the documented vulnerability may serve as cases supporting the need for future systematic changes and deeper additional analyses, not limited just to the library in question.

Preventing the single point of failure

The described problem would be mitigated if not a single but two or even more independent implementations were used to generate the RSA keypair. More generally, a secure multi-party protocol can be utilized to remove the single point of failure, not only during the keypair generation, but also during its use. The general goal is to provide tolerance against up to k out of t misbehaving (either faulty or intentionally malicious) participants [CDN05]. Multiple protocols based on common cryptographic primitives like RSA, Diffie-Hellman or Elliptic curve cryptography were proposed in literature [BF97; Gil99; Str03; Haz+12]. Such approaches protect not only against an intentionally malicious party, but also against unintentional mistakes weakening

the resulting key. The area of collaborative RSA keypair generation is well studied with the primary goal to generate parts (shares) of RSA keypairs, yet not to reveal the factorization of the resulting modulus N, until all or a specified number (threshold) of parties cooperate.

Gilboa's threshold RSA signature scheme [Gil99] requires collaboration during every signature operation, introducing protocol changes. A more efficient generation method by Straub based on 3-prime RSA [Str03] is not suitable for use by smartcards that implement offline signature generation with limited APIs, typically exposing only standard 2-prime RSA operations. Moreover, protocols securing against active adversaries, like that described in Hazay et al. [Haz+12], are time-consuming even on standard CPUs while having prohibitively long keypair generation phases on performance-limited hardware. Parsovs proposed a collaborative method that splits key generation between card manufacturer and cardholder [Par14]. The resulting 4-prime 4096-bit RSA key is generated from two 2048-bit parts during an interactive protocol executed before the card's first use, limiting the necessity to trust a vendor with the generation of the whole keypair, as well as removing the single point of failure.

Gennaro et al. proposed a distributed key generation algorithm for discrete-log cryptosystems (not directly applicable to RSA) [Gen+99], with extensions to provably secure distributed Schnorr signatures [SS01] and with the implementation shown to be efficient enough to run on cryptographic smartcards [Mav+17] as a mitigation of hardware Trojans.

Note that all the methods described above require changes to user interfaces and protocols and are therefore less suitable for legacy systems. However, a systematic adoption of secure multiparty protocols, instead of relying on a single vendor and implementation, can provide a significant overall increase of security of a system.

Analysis of other limited devices

The need for fast keypair generation on limited hardware naturally leads to a search for alternative methods for finding completely random primes. The generation method of Joye and Paillier [JP06] is one example. Therefore, other modifications (with respect to [JP06]) or completely different methods may have been adopted by other hard-

ware vendors. We did not detect any deviances in cards from 5 other manufacturers using our fingerprinting method. However, even a minor change to unit generation used in *RSALib* will suppress the bias that is detectable by our method (e.g., generators for p and q other than 65537), yet these changes will not automatically result in keys being secure against variations of our attack. The search for alternative detection techniques as well as attack variations represents possible future work.

4.5.3 Responsible disclosure

Disclosure of this vulnerability was made to *Manufacturer* in the beginning of February 2017 together with the tools demonstrating fingerprinting capabilities and practical factorization. The vulnerability was subsequently confirmed with further notification of the affected parties by *Manufacturer*.

We made public disclosure of the discovered issue in the middle of October 2017 together with the release a tool for fingerprint detection for provided public keys to facilitate a quick assessment of the presence of the vulnerability for end-users. The full details of the attack were published as [Nem+17b].

For the time being, we are not releasing our source code of the factorization algorithm. We believe that honest parties can make their own implementation based on our description.

4.6 Related work

The generation of RSA keys and attacks on them are the two main areas related to this work. Besides attacks on the messages (e.g., padding oracle [Ble98; Bor+09; Bro05] or related messages [YY05; Cop+96]), most attacks aim to deduce the private key from the corresponding public key. The attacks can be divided into two classes based on the assumptions about the key: 1) No additional information – methods such as Pollard p-1 [Pol74], Pollard Rho [Pol75; Bre80], and a class of several sieving methods (e.g., NFS, GNFS); 2) Partial information – low private or public exponent [Cop97; Wie90; BD99; BM03], implemen-

tation and side-channel attacks, and attacks based on Coppersmith's method [Cop96b].

The usage of generic attacks is limited to small RSA keys due to their exponential time complexity (the current record for a general 768-bit RSA [Kle+10] was broken using NFS). Only attacks from the second class are known to be used to break RSA moduli used in practice. Side-channel attacks (e.g., timing attacks, power analysis) are out of the scope of this work since they require active access to the device performing the RSA computation. Except for Wiener's attack [Wie90] for a small private exponent, other notable attacks belong to the same class as Coppersmith's attack.

In 2012, two independent teams [Hen+12; Len+12] analyzed RSA public keys on the Internet. The teams analyzed several millions of widespread keys in network devices such as keys in SSL certificates, SSH host keys and PGP keys. These teams observed that a small portion (0.5% of TLS, 1% of SSH) of public RSA keys shared prime factors. Due to insufficient entropy (e.g., SSL keys were generated by low-powered devices with no source of entropy) during the generation process, these keys can be trivially factorized using GCD. In 2013, Bernstein et al. [Ber+13] analyzed the "Citizen Digital Certificate" database of 3.2 million public RSA keys generated by smartcards used as the national IDs of Taiwanese citizens. In addition to recovering 184 keys that shared primes using a batch GCD computation, the authors adapted Coppersmith's algorithm and computed an additional 81 private keys. To our knowledge, this is the only practical application of Coppersmith's method to attack real RSA keys prior to our attack. Coppersmith's algorithm can be viewed as a universal tool for attacking RSA keys generated with improperly chosen parameters or originating from a faulty implementation. The algorithm was adapted for various scenarios where some bits of a factor, of the private exponent or of the message are known [BM06]. The factorization of moduli with known high [Cop96a] or low [Cop97] bits of a factor were among the first variants of the method. A nice overview of these methods can be found in [May09].

The generation of RSA keys is described in several standards (e.g., FIPS 186-4 [Nat13], IEEE 1363-2000 [IEE00] – see [LN11] for an overview), many having different requirements for the form of the primes. One feature is common to all these standards – the primes

should be generated randomly using a large amount of entropy. In addition to specialized construction methods (e.g., provable primes), the generation of RSA primes is typically performed in several iterations, repeating two fundamental steps: a random candidate is generated and then tested for primality. Since the primality test is a time-consuming process, several authors have proposed various speedups for the candidate generation process ([BDL93; Mau95; JPV00], see [JP06] for an overview of such methods). The current state of the art focused on constrained devices is described in [JP06], where the authors decreased the number of primality tests with a negligible loss of entropy (0.5 bits).

4.7 Conclusions

We presented a cautionary case of a vulnerable prime selection algorithm adopted in RSA key generation in a widely used security library of a cryptographic hardware manufacturer found in NIST FIPS 140-2 and CC EAL 5+ certified devices. Optimizations that were motivated by a higher performance in the key generation process have inadvertently led to significantly weakened security of the produced keys. The primes are constructed with a specific structure that makes the factorization of the resulting RSA keys of many lengths (including 1024 and 2048 bits) practically feasible with only the knowledge of the public modulus. Worse still, the keys carry a strong fingerprint, making them easily identifiable in the wild. The factorization method is based on our extension of the Howgrave-Graham refinement of Coppersmith's method.

To quantify and mitigate the impacts of this vulnerability, we investigated multiple domains where the RSA algorithm is deployed. Based on the specific structure of the primes, we devised a very fast algorithm to identify all vulnerable keys even in very large datasets, such as TLS or Certificate Transparency. Where public datasets were missing (eID, TPM, etc.), we attempted to collect some keys on our own. The results confirmed the use of the *RSALib* that produces vulnerable RSA keys across many domains.

There is mounting evidence that prime generation is a critical part of implementations that designers and developers struggle with.

Authoritative design notes for robust approaches should be produced and disseminated. Developers must follow existing standards without modifications.

Our work highlights the dangers of keeping the design secret and the implementation closed-source, even if both are thoroughly analyzed and certified by experts. The lack of public information causes a delay in the discovery of flaws (and hinders the process of checking for them), thereby increasing the number of already deployed and affected devices at the time of detection.

The certification process counter-intuitively "rewards" the secrecy of design by additional certification "points" when an implementation is difficult for potential attackers to obtain - thus favoring security by obscurity. Relevant certification bodies might want to reconsider such an approach in favor of open implementations and specifications. Secrecy may increase the difficulty of spotting a flaw (above the capability of some attackers) but may also increase the impacts of the flaw due to the later discovery thereof.

5 Amplification of TLS vulnerabilities on the Web

Implementing cryptographic functionality is not trivial, and mistakes can lead to severe vulnerabilities, as we saw in the previous chapter. It is especially true for a complex system with many complicated parts that can interact in unexpected ways, such as in the TLS protocol.

In this chapter, we focus on known practical cryptographic attacks on different building blocks of the TLS protocol. An important part of TLS is the RSA key exchange – the application of RSA asymmetric encryption to establish the keys used for symmetric encryption. It was repeatedly shown that developers struggle with that part of the implementation, likely due to the complicated nature of the padding scheme used since SSL, the early predecessor of the TLS protocol. Although we consider issues in other schemes as well, fittingly with the topic of the thesis, the implementation mistakes in RSA decryption are the most common reasons behind exploitable vulnerabilities.

We performed measurements that illustrate the idea of complex interactions. Unlike studies that evaluated the prevalence of vulnerabilities on popular domains, we also study the supporting infrastructure, such as related domains and servers that provide imported resources. Different servers that belong to the same organization may run on distinct implementations of TLS, some of which might be less secure than others. We detected that vulnerabilities get amplified due to the interconnected nature of the Web. Sometimes the principle of the weakest link applies. As an example, a single server that offers an exploitable RSA padding oracle undermines the security of all servers that use the same RSA key, even if they perform RSA decryption correctly. By considering such challenges, we contribute a more realistic estimate of the prevalence of exploitable vulnerabilities.

The results in this chapter were published in [Cal+19].

5.1 Introduction

The HTTP protocol is the central building block of the Web, yet it does not natively provide any confidentiality or integrity guarantee. HTTPS

protects network communication against eavesdropping and tampering by running HTTP on top of cryptographic protocols like Secure Socket Layer (SSL) and its successor Transport Layer Security (TLS), which allow for the establishment of encrypted bidirectional communication channels. Besides confidentiality and integrity, HTTPS also ensures authentication, because clients and servers may prove their identity by presenting certificates signed by a trusted certification authority. HTTPS has been increasingly recognized as a cornerstone of web application security over time and it is routinely employed by more and more websites, to the point that the average volume of encrypted web traffic has surpassed the average volume of unencrypted traffic according to data from Mozilla [Fel+17]. It is plausible to believe that, in a near future, HTTP will be (almost) entirely replaced by HTTPS, thanks to initiatives like Let's Encrypt and the actions taken by major browser vendors to mark HTTP as 'not secure' [Sch18].

Security experts know well that the adoption of HTTPS is necessary for web application security, but not sufficient. Web applications can be attacked at many different layers, for example on session management [Cal+17]. Moreover, the correct deployment of HTTPS itself is far from straightforward [Kro+17]. For instance, bad security practices like the lack of adoption of HTTP Strict Transport Security (HSTS) may allow attackers to sidestep HTTPS and completely void its security guarantees. But even when HTTPS is up and running, cryptographic flaws in SSL/TLS may undermine its intended security expectations. Many attacks against SSL/TLS have been found, allowing for information disclosure via side-channels or fully compromising the cryptographic keys used to protect communication [Adr+15; Avi+16; Beu+15; BL16a; BSY17; MDK14]. These attacks are not merely theoretical: they have been shown to be effective in the wild and open data from Qualys [Qua18] suggest that many servers are vulnerable to them. Several papers have also discussed the results of similar data collections [BSY17; DCE17; HFH16; Val+17; Val+18].

Despite this availability of data, however, previous analyses provide only a very limited picture of how much cryptographic weaknesses in HTTPS implementations harm the security of the current Web. First, these studies are based on large-scale detections of server-side vulnerabilities, but they do not provide a thorough account of their *exploitability on modern clients*. Many known vulnerabilities such

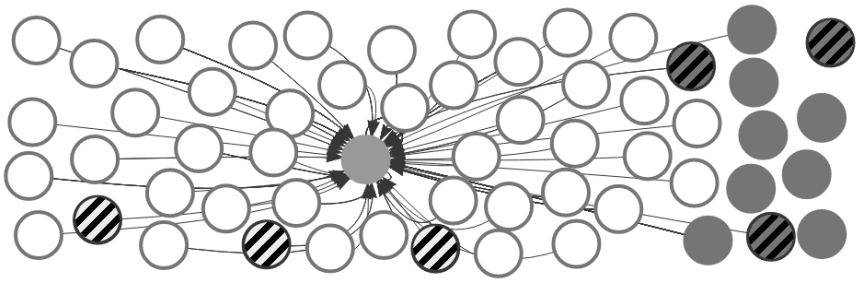

Figure 5.1: An anonymized top Alexa website (central circle) and its sub-domains (gray, on the right) and dependencies (white, with arrows). The website is entirely deployed over HTTPS, but becomes insecure due to three vulnerable sub-domains and three vulnerable dependencies (striped circles).

as Bleichenbacher's padding oracle attack on PKCS #1 v1.5 RSA encryption [Ble98] or various padding oracle attacks on Cipher Block Chaining (CBC) mode ciphers [AP13; MDK14; Vau02] rely on specific assumptions on both the client and the server to be exploited, such as that the TLS peers will negotiate a specific ciphersuite like RSA key exchange or use a symmetric cipher in CBC-mode, respectively. Hence, the mere existence of a vulnerability does not necessarily imply the possibility to attack a TLS connection between an up-to-date client and a vulnerable server, since all modern browsers implement various mitigations that prevent many of the known TLS attacks. Moreover, attacks against TLS at the transport layer may drastically differ in terms of their *impact at the application layer*: for example, the POODLE-TLS attack [Smi14] can gradually leak a secret, but it requires the attacker to force the browser into re-sending the secret many times. Thus, the attack can leak a session cookie by injecting requests from a page under the attacker's control, but not a password that is inserted by the user on a secure login page and only sent once.

In this chapter we present the first systematic quantitative evaluation of web application insecurity due to cryptographic HTTPS vulnerabilities. The analysis relies on a characterization of TLS vulnerabilities in terms of *attack trees* [Sch04] capturing the conditions for the various attacks to be enabled and on a crawl of the top 10,000 websites from Alexa supporting HTTPS, including all their dependencies (hosts

from which sub-resources are included) and sub-domains. Crawling dependencies and sub-domains is of ultimate importance, as secure websites might be broken by importing sub-resources or sending domain cookies over vulnerable TLS channels. The complexity of the web ecosystem, in fact, *amplifies* the effect of TLS vulnerabilities, as illustrated in Figure 5.1. Our results are disquieting:

- 898 websites are fully compromisable, allowing for script injection, while 977 websites present low integrity pages that the attacker can tamper with. Fully compromisable sites include e-commerce sites such as alibaba.com, e-banking services such as deutsche-bank.de and major websites such as myspace.com and verizon.com. 660 out of the 898 compromisable websites (73.5%) include external scripts from vulnerable hosts, thus empirically demonstrating that the complexity of web applications enormously amplifies their attack surface;

- 10% of the detected login forms have confidentiality issues, which may enable password theft. 412 websites may be subject to cookie theft, exposing to session hijacking, and 543 websites are subject to cookie integrity attacks. Interestingly, we found that more than 20% of the analyzed domain cookies can be potentially leaked, suggesting that the organization of web applications as related sub-domains amplifies their attack surface and needs to be carefully analyzed;

- 142 websites include content from vulnerable hosts of the popular tracker PubMatic and thus expose users to profiling attacks. Remarkably, this privacy attack can be amplified by the previous finding on compromisable websites, so as to affect up to 968 websites. This shows once more that attacks against TLS on external resources may expose otherwise secure websites to severe threats.

One of the original aspects of our work is that all of the presented attacks on web applications are exclusively due to practical TLS vulnerabilities that are enabled on the server and not prevented by modern browsers, thus potentially exploitable. Our findings show that a limited number of practical TLS vulnerabilities are amplified by

the web ecosystem and have a huge practical impact on otherwise secure websites that depend on or are related to the vulnerable hosts. We found vulnerabilities in popular, security-conscious websites. For example, because of TLS weaknesses in related hosts or dependencies, it is possible to break password confidentiality on myspace.com, session security on yandex.com and cookie integrity on live.com. We responsibly disclosed our findings to the interested websites.

Contributions and structure. In this work, we make the following contributions:

1. we review existing cryptographic attacks against TLS, identifying those which are still effective on modern clients. We then characterize such attacks in terms of attack trees, which identify conditions to break the confidentiality and/or integrity properties of the TLS protocol. To the best of our knowledge, this is the most systematic model of such attacks presented in the literature – with a special focus on their practical impact – and can serve other security researchers working in the area (Section 5.3);

2. we build an analysis platform which implements the checks defined by the attack trees and we run it on the homepages of the top 10,000 websites of the Alexa ranking supporting HTTPS. As part of this data collection process, we also scan 90,816 hosts which either (i) store sub-resources included in the crawled pages or (ii) are sub-domains of the websites. These hosts have a major impact on the security of the crawled websites, which we precisely assess (Section 5.4);

3. we rigorously identify a number of severe web application attacks enabled by vulnerable TLS implementations and we run automated checks for them on the collected data. We focus on three different aspects of web application security: *page integrity* (Section 5.5), *authentication credentials* (Section 5.6) and *web tracking* (Section 5.7). This list is not meant to be exhaustive, yet it is rich enough to cover important security implications of existing cryptographic flaws of TLS on major websites.

Section 5.2 provides background on TLS and Section 5.8 provides our closing perspective, discussing related work, ethical issues and limitations of our study. Finally, Section 5.9 presents some of our additional findings.

5.2 Background on TLS

In this section, we describe TLS 1.0, 1.1 and 1.2. Readers who are already familiar with TLS can safely skip this section. We do not discuss TLS 1.3 [Res18], as there are no known attacks against it due to the removal of vulnerable cryptographic constructions used in previous protocol versions [Res18, Section 1.2]. Notice that version 1.3 is not yet widely supported in the wild: only 5.2% of hosts in our scan supported some draft version of TLS 1.3 (the final version was not yet published at the time of the scan). Moreover, we do not discuss certificate-based client authentication as it is rarely adopted on the Web.

The TLS protocol consists of the following sub-protocols:

- **Record Protocol** carries the data, that are optionally encrypted and authenticated, of the application data protocol and the remaining TLS sub-protocols;
- **Handshake Protocol** negotiates cryptographic keys and authenticates the server;
- **Change Cipher Spec Protocol** signals to the other peer that the subsequent records will be encrypted and authenticated under the negotiated keys;
- **Alert Protocol** signals status changes, with warnings and terminating fatal alerts, following e.g., decryption errors.

5.2.1 The Handshake Protocol

We describe in detail the Handshake Protocol, as it is the one responsible for agreeing on the cryptographic algorithms and keys used to protect messages and for authenticating the server. As such, it constitutes a clearly sensitive target for network attackers. The Handshake

Protocol is an authenticated key exchange protocol. The peers negotiate the TLS version and the cryptographic algorithms (ciphersuites) for key exchange, server authentication, and Record Protocol protection.

The client initiates the handshake with a `ClientHello` message, that includes the highest supported TLS protocol version, a random nonce for key derivation, the session identifier, the list of supported ciphersuites, the supported compression methods (usually empty, as TLS compression is deprecated for security reasons), and optional TLS extensions.

The server responds with a `ServerHello` message with the lower between its highest supported protocol version and the client's version, a random nonce, the session identifier, the selected ciphersuite and compression method, and selected extensions (a subset of those offered by the client). The server should follow an ordering of the ciphersuites, ideally selecting the most secure ciphersuite offered by the client. If there are no supported algorithms in common, the server responds with a handshake failure alert.

The server also sends its X.509 certificate in the `Certificate` message, that links its identity to its public key. Depending on the selected ciphersuite, it may send a `ServerKeyExchange` message contributing to the key material. The client sends the `ClientKeyExchange` message with its key material. The shared key material is called the Premaster Secret (PMS) and is used together with the exchanged random nonces to compute the Master Secret, which is in turn used to derive the session keys for the Record Protocol. Once the Master Secret is shared, the peers run the Change Cipher Spec Protocol and start protecting their messages.

Finally, the client and the server mutually exchange the `Finished` message containing a transcript of the handshake. If the peers received different messages, possibly due to tampering by an attacker, their transcripts will differ. Since the communication is encrypted and authenticated with the session keys at this point, the attacker cannot tamper with the transcripts. The PMS is shared using a public key that is tied to the identity of the server, hence the server authenticates by using the PMS to compute the session keys.

5.2.2 Ciphersuites

A key ingredient of the Handshake Protocol is the negotiation of the cryptographic mechanisms in the ciphersuite. The most common algorithms are:

- **Key exchange:** how to share the PMS:
  - **RSA key exchange:** the client randomly generates a PMS, encrypts it with the RSA public key of the server obtained from the server's trusted certificate, and sends it in the `ClientKeyExchange`;
  - **Static Diffie-Hellman key exchange – (EC)DH:** the DH parameters are defined either on a prime field (DH) or on an elliptic curve (ECDH). The client generates a random (EC)DH key and sends the public part in the `ClientKeyExchange`. The public key of the server is contained within its certificate. The shared DH secret is used as the PMS;
  - **Ephemeral Diffie-Hellman key exchange – (EC)DHE:** similar to the previous case, however the client and the server generate fresh (ephemeral) (EC)DHE keys and send them in the `Client-` and `Server- KeyExchange` messages, respectively. The server must sign its message with a private key corresponding to its certificate. DHE uses RSA or DSA [Nat13], ECDHE uses RSA or ECDSA [Nat13].
- **Confidentiality and integrity:** how messages sent over the Record Protocol are protected:
  - **Block ciphers in AEAD mode:** Authenticated Encryption with Associated Data (AEAD) combines encryption and authentication in a single primitive. Examples are AES in the GCM or CCM mode of operation;
  - **Block ciphers in CBC mode with MAC:** combination of CBC mode of operation of a symmetric block cipher with Keyed-hash Message Authentication Code (HMAC) for authentication. The order of operations is MAC-then-Pad-then-Encrypt. For example, AES, Camellia, Triple-DES or

DES in CBC mode combined with HMAC based on SHA-2, SHA-1 or MD5;

– **Stream cipher with MAC:** for example, ChaCha20 with Poly1305 (that combine into an AEAD primitive) or RC4 with HMAC based on SHA-1 or MD5.

5.3 Attack trees for TLS security

We describe notable cryptographic attacks against TLS and divide them by their impact on confidentiality and integrity of the communication. We discuss how the attacks are mitigated by client configuration and specific countermeasures, focusing on attacks that fall under our threat model. See Section 5.9.1 for out of scope attacks and Section 5.9.2 for more details on the attacks introduced in this section.

5.3.1 Threat model

We assume an active network attacker able to add, remove or modify messages sent between a client and a server. The attacker also controls a malicious website, say at `evil.com`, which is navigated by the attacked client. By means of the website, the attacker can inject scripts in the client from an attacker-controlled origin, which is relevant for a subset of the attacks. However, the attacker can neither break the Same Origin Policy (SOP)[1] nor exploit any bug in the browser. We assume the attacker cannot exploit timing side-channels, since the feasibility of such attacks is generally hard to assess.

The client is a modern browser that (i) supports TLS 1.0, 1.1, and 1.2 with key establishment based on ECDH and AEAD ciphersuites (cf. MozillaWiki [Veh18] for the purpose of "Modern" compatibility); (ii) does not support SSLv3 or lower, does not offer weak or anonymous ciphersuites (such as DES, RC4 and EXPORT ciphers, or suites without encryption or authentication) and enforces a minimal key size of cryptographic algorithms; (iii) correctly handles certificate validation and rejects certificates with weak algorithms. All the major browsers released in the last two years satisfy these assumptions, starting from

1. `https://developer.mozilla.org/docs/Web/Security/Same-origin_policy`

Firefox 44, Chrome 48, IE 11 on Windows 7, Edge, Opera 35, Safari 10, and Android 6.0.

5.3.2 Review of known attacks against TLS

Protocol version downgrade

A TLS server should respond to a `ClientHello` with the offered version of the protocol, or the highest it supports. However, some legacy servers simply drop connections with unsupported TLS versions, without offering an alternative. Thus, browsers may repeat the handshake with a lower protocol version. An attacker in the middle could drop `ClientHello` messages until the client downgrades to an older, vulnerable version of the protocol. To prevent this attack, the client attaches a fake ciphersuite to repeated handshake attempts, as defined in RFC 7507 [ML15], indicating that the handshake did not use the highest client-supported TLS version. The presence of that ciphersuite in a `ClientHello`, with a TLS version that is lower than the highest supported by the server, reveals a potential attack and should be treated as such by the server. Safari, Internet Explorer, and Edge fall back to TLS 1.0. Only Safari appends the ciphersuite. Firefox, Chrome, and Opera, instead, removed insecure fallback entirely when the `ClientHello` messages are dropped.

RSA decryption oracles

In the RSA key exchange, the client chooses the PMS and sends it to the server, encrypted under the server's public RSA key. TLS uses the padding scheme defined in PKCS #1 v1.5 [Kal98], which is known to be vulnerable to a padding oracle attack [Ble98]. The attack is possible when the server provides a padding oracle, i.e., when it behaves differently when decrypting messages that have invalid paddings. An attacker can multiply a ciphertext to create a new ciphertext (RSA is malleable), until a new correctly padded message is forged. When this happens, the attacker learns partial information about the plaintext message and the process can be iterated until the key exchange is fully decrypted. The original attack was proposed by Bleichenbacher in 1998 [Ble98] and requires on the order of million connections to decrypt a ciphertext. The attack was later improved [Bar+12; JSS15;

KPR03; Mey+14], especially in the presence of an oracle that does not strictly enforce the padding scheme [Bar+12], to require on the order of tens of thousands of messages. In our analysis, we only consider such strong version of the oracle as exploitable.

RSA signature oracles

A very fast decryption oracle can be used to compute RSA signatures. Hence, even without the knowledge of the private key, an attacker can impersonate the server in the (EC)DHE exchange with such oracle. The attack applies to all TLS versions up to TLS 1.2. However, the signature generation using a Bleichenbacher's oracle is even slower than the decryption [BSY17]. Therefore, the attacker would prefer the decryption of RSA key exchange, if supported by the targeted host. Interestingly, a signature oracle makes it possible to impersonate the target server even with other certificates valid for that target (such as wildcard certificates).

Advanced RSA padding oracles – DROWN and key reuse

When a server is vulnerable to the decryption oracle, all servers that use the same RSA key for key encryption (e.g., due to using the same certificate) are vulnerable to the decryption of the key exchange, even if they do not provide the oracle directly. Furthermore, TLS can be enabled for other application level protocols than HTTPS, such as email (SMTP, POP3, and IMAP with STARTTLS, or SMTPS, IMAPS, POP3S). The attack surface of the DROWN attack [Avi+16] was in fact amplified by the possibility of using vulnerable servers supporting SSLv2 in order to break servers running newer protocol versions. DROWN uses the fact that SSLv2 provides the padding oracle in combination with weak export grade ciphersuites and specific OpenSSL bugs. The attack comes in two variants, General and Special, requiring respectively about 8 hours and less than a minute to complete. Thus, only the Special case is suitable for Man In The Middle (MITM) attacks. Not all handshakes are vulnerable: 1 out of 900, for the General case, and 1 out of 260 for the Special case.

RSA padding oracle countermeasures

TLS 1.0 [DA99], 1.1 [DR06], and 1.2 [DR08] introduced countermeasures to remove the padding oracle, instead of replacing the padding scheme. However, the ROBOT attack [BSY17] has shown that a surprisingly high number of implementations in the wild still present padding oracles that can be used to decrypt RSA encrypted messages. The attacks are partially mitigated by the support for Perfect Forward Secrecy, typically by preferring the elliptic curve Diffie-Hellman key establishment with ephemeral private keys (ECDHE) over the RSA key exchange on the server side. Since all modern web browsers support ECDHE cipher suites [Veh18], the RSA key exchange will be voluntarily negotiated only with servers that prefer it due to lack of ECDHE support or bad configuration. It would be thus recommended to completely disable RSA encryption at the server side [BSY17].

CBC mode padding oracles

TLS uses the CBC mode of operation of a symmetric block cipher with MAC-then-Pad-then-Encrypt scheme for record-level encryption. Since the padding is not covered by the MAC, changing the padding does not change the integrity of the message, and could enable a padding oracle vulnerability. A class of vulnerabilities of the MAC-then-Pad-then-Encrypt construction was described by Vaudenay [Vau02] and Canvel et al. [Can+03]. The attacks are based on distinguishing failures due to bad padding and due to failed integrity check. In TLS, the server should issue the same response in both situations, however there are buggy implementations (e.g., [Som16a]) that produce different errors. The POODLE attack [MDK14] leverages the above padding oracle problem in combination with the fact that SSLv3 (and some flawed TLS implementations) only checks the last byte of padding. Since a padding error ends in a termination of the session, the attacker must be able to force the client to open a new session every time she wants to make a guess. Furthermore, the client must repeat the target secret s in every connection, e.g., when s is a secret cookie attached to every HTTPS request. All CBC attacks can be mitigated in TLS 1.2 by supporting either AEAD ciphersuites or stream ciphers that do not require padding, on both servers and clients (as in modern

browsers). TLS version downgrades must also be mitigated, to prevent a downgrade to a version that only supports CBC-mode ciphers.

Heartbleed

Due to memory management problems in server implementations, an attacker could reveal the long-term private keys of the server, thus allowing a full impersonation of the server [Syn14; Dur+14].

5.3.3 Insecure channels

To understand the impact of cryptographic flaws of TLS on web application security, it is useful to categorize known cryptographic attacks in terms of the security properties they break. We propose three categories of insecure channels:

- **Leaky:** a channel established with servers vulnerable to confidentiality attacks, which give the attacker the ability to decrypt all the network traffic (Section 5.3.4);
- **Tainted:** a channel susceptible to Man In The Middle (MITM) attacks, which give the attacker the ability to decrypt and arbitrarily modify all the network traffic (Section 5.3.5). Tainted channels are also leaky;
- **Partially leaky:** a channel exposing side-channels which give the attacker the ability to disclose selected (small) secrets over time. These channels typically rely on a *secret repetition* assumption, because the attacker abuses the exchange of repeated messages containing the secret on the vulnerable channel (Section 5.3.6). Leaky and tainted channels also qualify as partially leaky.

In the rest of this section, we precisely characterize how we mapped existing cryptographic attacks against TLS to the proposed channel categories in terms of attack trees.

5.3.4 Leaky channels

Channels are *leaky* when established with servers vulnerable to attacks that fully compromise confidentiality. The attacker tries to obtain the

```
GOAL Learn the session keys (allows decryption)
| 1 Decrypt RSA key exchange after the handshake
  & 1 RSA key exchange is used
    | 1 RSA key exchange is preferred in the
        highest supported version of TLS
    | 2 Downgrade is possible to a version of TLS
        where RSA key exchange is preferred
  & 2 RSA decryption oracle (DROWN or Strong
      Bleichenbacher's oracle) is available on:
    | 1 This host
    | 2 Another host with the same certificate
    | 3 Another host with the same public RSA key
```

Figure 5.2: Attack tree for leaky channels.

PMS to learn the session keys, giving her the ability to decrypt all the captured network traffic.

Figure 5.2 shows the attack tree of conditions that enable the attacker to learn the session keys. The main goal is listed on the first line. Each goal or sub-goal may have alternative ways of reaching it (marked as logical OR '|') or it may require several sub-goals to be valid at once (marked as logical AND '&'). Sub-goals are differentiated from their parent goal by increased indentation. Leaves, i.e., goals without sub-goals, evaluate to True or False based on a concrete test (e.g., for the presence of a vulnerability), a detected server configuration, or are the result of a stand-alone, separate tree. If the entire tree evaluates to True, the host suffers from an exploitable vulnerability that can facilitate the main goal.

The attacker may obtain the PMS by decrypting the key exchange (1). The parties must use RSA key exchange (1.1). Hence, the client must support it and the server must prefer it either in the highest version of TLS supported by both parties (1.1.1), or in any other commonly supported version, if protocol version downgrade is not properly mitigated (1.1.2). The attacker decrypts the RSA key exchange (1.2) either using Strong Bleichenbacher's oracle [BSY17] or with the DROWN attack [Avi+16]. The oracle could be present on the target host directly (1.2.1), or on a different host that uses the same certificate (1.2.2) or at least the same RSA key (1.2.3).

```
GOAL Potential MITM (decryption and modification)
| 1 Force RSA key exchange by modifying ClientHello
    and decrypt it before the handshake times out
  & 1 RSA key exchange support in any TLS version
  & 2 Fast RSA decryption oracle (Special DROWN or
      Strong Bleichenbacher's oracle) available on:
    | 1 This host
    | 2 Another host with the same certificate
    | 3 Another host with the same public RSA key
| 2 Learn the session keys of a long lived session
  & 1 Learn the session keys (Figure 5.2)
  & 2 Client resumes the session
    | 1 Session resumption with tickets
    | 2 Session resumption with session IDs
| 3 Forge an RSA signature in the key establishment
  & 1 Fast RSA signature oracle (Strong
      Bleichenbacher's oracle) is available on:
    | 1 This host
    | 2 Another host with the same certificate
    | 3 Another host with the same public RSA key
    | 4 A host with a certificate where the Subject
        Alternative Names (SAN) match this host
  & 2 The same RSA key is used for RSA key exchange
      and RSA signature in ECDHE key establishment
| 4 Private key leak due to the Heartbleed bug
```

Figure 5.3: Attack tree for tainted channels.

5.3.5 Tainted channels

Channels are *tainted* if the attacker can mount a MITM attack that gives her the ability to decrypt and modify all the traffic between the server and the client. Hence, tainted channels are also leaky (implied by the ability to decrypt). The attacker must learn the PMS of an active session or she must influence its value and successfully impersonate the server. The attack tree is shown in Figure 5.3 and described below.

Obtaining the PMS early is characteristic of a tainted channel. If the attacker learns the session key after the session is over, she can only decrypt, and the channel is only considered to be leaky. Such a channel can be upgraded back to tainted in case of session resumption.

The attacker can force the use of RSA key exchange by modifying the `ClientHello` sent to the server to only contain such ciphersuites (1). Naturally, the server must support such ciphersuite (1.1). The modification leads to different handshake transcripts, hence the decryption of the key exchange must be performed very fast, in order to generate valid `Finished` messages before the peers time out. Hence, the attacker needs access to a fast instantiation of Strong Bleichenbacher's oracle [BSY17] or to a server vulnerable to the Special variant of the DROWN attack [Avi+16] (1.2). The authors of the ROBOT attack [BSY17] estimate that it should be feasible to decrypt the key exchange fast enough (in a few seconds) if the attacker can parallelize the requests across multiple servers of the attacker and the target. An analysis of such parallel attack was done by Ronen et al. [Ron+19].

Alternatively, the attacker may gain more time to obtain the session keys, if they are long lived (minutes to hours) (2). She captures an RSA key exchange and decrypts it offline (2.1), through the techniques of Section 5.3.4 (Figure 5.2) as she cannot modify the initial `ClientHello` at will. She then intercepts a resumed session with full MITM capabilities (2.2). Server may support session resumption without server-side state (2.2.1) [Sal+08] or with server-side state (2.2.2) [DR08].

Under some conditions, a very efficient RSA decryption oracle can be used to forge signatures (3). The oracle can be found on a variety of hosts (3.1.1 – 3.1.3). Additionally, a host can be attacked using a certificate that it neither uses nor shares an RSA key with, if the host appears on the certificate's list of Subject Alternative Names (SAN) (3.1.4). The certificate's RSA key used for signing (EC)DHE parameters

```
GOAL Partial decryption of messages sent by Client
| 1 CBC padding oracle on the server
  | 1 POODLE-TLS padding oracle
    & 1 Server checks TLS padding as in SSLv3
    & 2 Any vulnerable CBC mode ciphersuite is used
      | 1 A CBC mode ciphersuite is preferred
          in the highest supported version of TLS
      | 2 Downgrade is possible to a version of TLS
          where a CBC mode ciphersuite is preferred
  | 2 CBC padding oracle - OpenSSL AES-NI bug
    & 1 Server is vulnerable to CVE-2016-2107
    & 2 A ciphersuite with AES in CBC mode is used
      | 1 AES in CBC mode is preferred in the
          highest supported TLS version
      | 2 Downgrade is possible to a TLS version
          where AES in CBC mode is preferred
```

Figure 5.4: Attack tree for partially leaky channels.

must be the same as the RSA key used for RSA key exchange by a server with a decryption oracle (3.2).

Finally, the attacker might obtain the private key of the server due to the Heartbleed memory disclosure bug (4) [Syn14]. For ethical reasons, we did not attempt to extract the private keys when we detected Heartbleed, yet it was reliably shown possible [Ind14].

5.3.6 Partially leaky channels

Channels are partially leaky if they allow for a partial confidentiality compromise of secrets sent by the client to the server. Leaky and tainted channels are also partially leaky. The conditions are described by the attack tree in Figure 5.4. To exploit a CBC padding oracle (1), the attacker must force repeated requests containing the secret (*secret repetition*) and she is required to partially control the plaintext sent by the client to a vulnerable server, e.g., by modifying the URL in the header of the request. We check the server for the presence of two CBC padding oracle types (as explained in Section 5.3.2). They are instantiated as the TLS version of the POODLE attack [Smi14; MDK14] (1.1) due to incorrect padding checks (1.1.1) and as a buggy implementation [Som16a] providing a Vaudenay CBC padding oracle

[Vau02] (1.2) when using hardware accelerated AES (AES-NI) in certain versions of OpenSSL (1.2.1). Both attack types require the server to choose a vulnerable ciphersuite (1.1.2, 1.2.2). It could be chosen by the server in the highest TLS version (1.1.2.1, 1.2.2.1) or following a protocol version downgrade (1.1.2.2, 1.2.2.2).

5.4 Experimental setup

We developed an analysis platform to identify exploitable cryptographic weaknesses in TLS implementations and estimate their import on web application security. The platform employs a crawler to perform a vulnerability scan of the target website, testing also hosts which either store sub-resources included by the homepage or belong to related domains. Confidentiality and integrity threats are identified by matching the relevant conditions of the attack trees introduced in Section 5.3 against the output of existing analysis tools.

5.4.1 Analysis platform

The analysis platform performs the following steps: (i) access the website, such as `example.com`, by instrumenting Headless Chrome with Puppeteer;[2] (ii) collect the DOM of the page at `example.com`, along with its set of cookies and the hosts serving sub-resources (such as scripts, images, stylesheets and fonts) included by the page; (iii) enumerate the sub-domains of `example.com` by querying the Certificate Transparency[3] logs and by testing for the existence of common sub–domains, such as `mail.example.com`; (iv) run existing analysis tools to identify cryptographic vulnerabilities on the target website and on all the hosts collected in the previous steps; (v) map the output of the tools to the conditions of the attack trees to find exploitable vulnerabilities.

The analysis tools include testssl.sh,[4] TLS-Attacker [Som16b] and the nmap plugin for Special DROWN,[5] which combined provide

2. `https://github.com/GoogleChrome/puppeteer`
3. `https://www.certificate-transparency.org/`
4. `https://github.com/drwetter/testssl.sh`
5. `https://nmap.org/nsedoc/scripts/sslv2-drown.html`

Insecure channel	*Attack*	*Attack tree reference*		*Vulnerable hosts*
Leaky	Decrypt RSA key exchange offline	(1)	Figure 5.2	733
Tainted	Force RSA key exchange and decrypt it online	(1)	Figure 5.3	1,877
	Learn the session keys of a long lived session	(2)		615
	Forge an RSA signature in the key establishment	(3)		2,279
	Private key leak due to the Heartbleed bug	(4)		47
Partially leaky	POODLE-TLS padding oracle	(1.1)	Figure 5.4	816
	CBC padding oracle – OpenSSL AES-NI bug	(1.2)		96

Table 5.1: Overview of the detected insecure channels.

enough information. For ethical reasons, we did not perform any aggressive testing for the presence of oracles other than the checks run by these tools, e.g., we did not evaluate the performance of servers with respect to the number of oracle queries they can answer in a short time. Still, if some untested conditions have been considered realistic in the literature, e.g., the performance of a Strong Bleichenbacher's Oracle for online decryption or for signature computation [BSY17], we report the vulnerability as exploitable.

5.4.2 Data collection and findings

We used our analysis platform to collect data from the Alexa top 1M list retrieved on July 20, 2018. We scanned websites starting at the top of the ranking until we collected 10,000 websites correctly served over HTTPS. Their sub-resources and related domains added up to 90,816 more hosts that underwent a vulnerability analysis, completed at the beginning of August 2018.

Our tool reported exploitable TLS vulnerabilities in 5,574 hosts (5.5%). 4,818 hosts allow for the establishment of tainted channels, which is the most severe security threat. 733 hosts allow for the estab-

lishment of leaky channels, while 912 allow for partially leaky channels. The majority of vulnerabilities is due to the 20 years old Bleichenbacher's attack [Ble98] and its newest improvement ROBOT [BSY17]. Only 6.5% of the scanned hosts actually prefer RSA key exchange in their highest supported TLS version, yet 76.9% hosts support it, presumably to maintain backward compatibility with old clients. More than 90% of servers support a key exchange that provides Perfect Forward Secrecy. Hence, the majority of the exploitable hosts could be secured by stopping the support for RSA key exhange. We provide a breakdown of the identified insecure channels in Table 5.4.1 and we comment it below.

Leaky channels

The connections to 733 hosts could be decrypted using ROBOT or DROWN after the attacker captured the traffic – goal (1) of Figure 5.2. 727 hosts preferred the RSA key exchange (1.1.1), hence no action would be necessary to make the peers negotiate RSA. Only on 6 hosts the attacker would need to use the protocol version downgrade to force the usage of RSA key exchange (1.1.2) instead of Diffie-Hellman (DH). We found 136 hosts vulnerable to ROBOT that used ECDHE in their highest protocol version and properly implemented protocol version downgrade mitigation, showing the importance of the countermeasure. Out of the 733 vulnerable hosts, 592 hosts were directly exploitable (1.2.1), while 141 were only exploitable due to sharing a certificate (1.2.2) or an RSA key (1.2.3) with a vulnerable host. Hence, a conventional tool that only checks the host directly for the presence of ROBOT would not detect confidentiality problems on 19% of the exploitable hosts.

Tainted channels

In total, 4,818 hosts made connections over tainted channels due to MITM attacks (Figure 5.3). 615 hosts were exploitable due to the compromise of a resumed session (2), where the attacker can decrypt the key exchange over a longer period. 1,877 additional hosts were susceptible to online RSA key exchange decryption attacks (1). The attack was also possible for the previously mentioned 615 hosts, without

relying on the client to resume the session (2.2), yet requiring a faster computation (1.2). When a decryption oracle is available on a host, each certificate that uses the same RSA key for signatures could be used to impersonate all the hosts that appear in its Subject Alternative Name extension (SAN) (3). We found 2,279 such hosts, that could not be impersonated with a less demanding version of the MITM attack: (1) or (2). It is worth noticing that only 1,893 hosts in our scan had a strong ROBOT oracle, yet the number of exploitable servers due to ROBOT is much higher. This shows that the sharing of certificates and RSA public keys, as well as the list of hostnames in the SAN extension, should be kept minimal. Luckily, only 47 hosts were vulnerable to Heartbleed (4). When a private RSA key is extracted in this way, the attacker can repeatedly impersonate the host without its involvement.

Partially leaky channels

Exploitable partially leaky channels (Figure 5.4) were found on 912 hosts. Out of the 816 hosts with an exploitable POODLE-TLS padding oracle (1.1), 797 hosts preferred the vulnerable ciphersuite (1.1.2.1) and additional 19 hosts could be exploited after being downgraded to an older version of TLS due to a lack of protection from downgrades (1.1.2.2). Out of the 96 hosts with an exploitable OpenSSL AES-NI padding oracle (1.2), only 20 hosts were vulnerable in the preferred TLS version (1.2.2.1) and additional 76 hosts could be exploited after an unmitigated version downgrade (1.2.2.2). Other 68 hosts have been found affected by POODLE-TLS and 2 exposed OpenSSL AES-NI padding oracle, yet a modern browser would negotiate a more secure cipher making the vulnerabilities non-exploitable.

5.4.3 Roadmap

The presence of so many insecure channels is concerning, but their actual impact on web application security is unclear. In the rest of the chapter, we investigate and quantify this delicate point by focusing on selected aspects of web application security. Since we are interested in cryptographic attacks against HTTPS, we stipulate that every time we refer to *pages* / *channels* we implicitly refer to HTTPS pages / channels,

unless otherwise specified. Attacks enabled by the (partial) adoption of HTTP are out of the scope of this study.

5.5 Page integrity

In this section, we describe a number of attacks enabled by the presence of tainted channels, whose security impact ranges from content injection to SOP bypasses.

5.5.1 Security analysis

If a web page is received from a tainted channel, the attacker may be able to arbitrarily corrupt its contents, thus completely undermining its integrity guarantees. Moreover, even if the page was received from an untainted channel, the subsequent inclusion of scripts sent over tainted channels in the top-level document may fully compromise integrity. The only protection mechanism available in modern browsers against the latter threat is Subresource Integrity (SRI) [Akh+16], a relatively recent web standard which allows websites to bind to `<script>` tags an `integrity` attribute storing a cryptographic hash of the script which is expected to be included by them. If the included script does not match the hash, the script is not executed, so SRI can be used to prevent the threats of script injection via network attacks.

The two integrity attacks above are equally dangerous and the most severe ones in terms of security, because they grant to the attacker active scripting capabilities on the web page, which we can thus deem as *compromisable*.

Definition 1 (Compromisable Page)**.** *A page is* compromisable *if and only if any of the following conditions holds:*

1. *the page is received from a tainted channel;*
2. *the page includes scripts in the top-level document from tainted channels without using SRI.*

Notice that the definition does not refer to Content Security Policy (CSP) [Wes18], a web standard which can be used to prevent the execution of inline scripts and restrict content inclusion on web pages

by means of a white-listing mechanism. In fact, CSP is ineffective against network attackers: if a page is compromisable because it is received from a tainted channel, the attacker may just strip away the CSP headers and `<meta>` tags to disable the protection; if instead a page is compromisable because it includes scripts from tainted channels, observe that CSP does not prevent the replacement of legitimate scripts with arbitrary malicious contents.

A second class of threats we are interested in allows SOP bypasses through compromisable pages. If a host contains at least one compromisable page, SOP becomes largely ineffective at defending it, because the attacker may get active scripting capabilities in its web origin and get access e.g., to its cookies and web storage. This motivates the following definition.

Definition 2 (Compromisable Host)**.** *A host is* compromisable *iff it is possible to retrieve a compromisable page from it.*

Finally, besides these obvious threats, it is worth noticing that there are also other integrity attacks which are subtler than script injection, but may achieve results as severe as page compromise under specific circumstances. For example: (i) the inclusion of stylesheets and web fonts can be used to perform *scriptless attacks*, which may enable the exfiltration of confidential information stored in the DOM [Hei+14]; (ii) the inclusion of Scalable Vector Graphics (SVG) images using tags like `<embed>` may lead to the injection of malicious HTML and JavaScript contents [Hei+11]; (iii) the inclusion of iframes can lead to exploitations against the top-level document via the postMessage API [SS13]; (iv) the result of an XMLHttpRequest can be passed to a function like `eval`, which converts strings into executable code and thus enables script injection [Wei+16].

To comprehensively characterize the pages suffering from these potential integrity issues, we leverage the Mixed Content [Wes16] specification, which defines the reference security policy for the inclusion in HTTPS pages of contents delivered over HTTP channels. The key idea to uniformly capture these attacks is to reuse the definition of *blockable request* introduced in the Mixed Content specification, which mandates that compliant browsers must prevent HTTPS pages from sending this type of requests over HTTP channels.

Definition 3 (Blockable Request)**.** *A request is* blockable *if and only if it is not requesting any of the following resources:*

1. *images loaded via* `<img>` *or CSS;*
2. *video loaded via* `<video>` *and* `<source>`*;*
3. *audio loaded via* `<audio>` *and* `<source>`*.*

We similarly consider blockable requests over tainted channels as a possible source of integrity attacks, which leads to the following definition of *low integrity* page.

Definition 4 (Low Integrity Page)**.** *A page has* low integrity *if and only if any of the following conditions holds:*

1. *the page is compromisable;*
2. *the page includes sub-resources (other than scripts) via blockable requests sent over tainted channels.*

Low integrity pages which only satisfy the second condition do not necessarily provide active scripting capabilities to the attacker, yet they might still pose significant security threats in specific scenarios. That said, in the next sections we will often reason about the integrity of web pages to characterize additional web application attacks and our analysis will always be *optimistic*, i.e., we will assume that the attacker gets active scripting capabilities only in compromisable web pages and not in low integrity pages. We will also dispense with potential information leakages enabled by scriptless attacks [Hei+14], because they are not easy to exploit and depend on the details of specific web technologies. This conservative approach will limit the number of false positives in our security analysis.

5.5.2 Experimental results

The homepages of the 10,000 crawled websites included sub-resources from 32,642 hosts. Our analysis exposed 977 low integrity pages (9.8%), including 898 compromisable pages where an attacker can get active scripting capabilities. Examples of major security-sensitive websites whose homepage was found compromisable include e-shops

(alibaba.com, aliexpress.com, tmall.com), online banks (bankia.es, deutsche-bank.de, sparkasse.at, icicibank.com), social networks (myspace.com, linkedin.com, last.fm) and other prominent services (verizon.com, webex.com, livejournal.com).

Out of 898 compromisable pages, there are 238 pages received from tainted channels and 660 pages including scripts from tainted channels. Although the security dangers of these two cases are the same, the latter cases are particularly intriguing, because they show that the majority of the compromisable pages (73.5%) is harmed by the inclusion of external scripts. Since the majority of these scripts is hosted on domains which are not under the direct control of the embedding pages, SRI is the way to go to mitigate their threats: unfortunately, SRI is only used in 329 pages (3.3%) and does not prevent any page compromise in our dataset. Rather, we observe that there are 25 pages using SRI on some script tags, but are still compromisable because SRI is not deployed on *all* the script tags including contents from tainted channels.

Based on the previous considerations on external scripts, it is noteworthy that there exist popular script providers which are deployed on top of vulnerable HTTPS implementations, thus severely harming the integrity of a very large number of websites which include contents from them. Table 5.2 reports the most popular script providers which allow for the establishment of tainted channels, along with the number of the Alexa websites which include at least one script from them in their top-level document. These numbers show that by targeting only a couple of carefully chosen hosts, an attacker can fully undermine the integrity of a much larger number of websites, thus making integrity attacks cost-effective. For instance, consider the *LinkedIn Insight Tag*, a JavaScript code that enables the collection of visitors' data on webpages which include it and provides web analytics for LinkedIn ad campaigns. The script is loaded from a tainted channel served on snap.licdn.com (second row of Table 5.2), which is vulnerable to MITM attacks due to a host affected by ROBOT at rewards.wholefoodsmarket.com, that presents a valid certificate for snap.licdn.com. The inclusion of this script threatens the integrity of 126 websites among the ones we analyzed, including notable examples such as auth0.com, britishairways.com, linode.com and teamviewer.com.

Script Provider	*Including Websites*
hm.baidu.com	188
snap.licdn.com	126
ads.pubmatic.com	47
zz.bdstatic.com	39
cdn.tagcommander.com	37
tag.baidu.com	20
geid.wbtrk.net	19
cdn.wbtrk.net	19
cdn.blueconic.net	14
dup.baidustatic.com	12

Table 5.2: Top script providers introducing integrity flaws.

5.6 Authentication credentials

In this section, we discuss the impact of (partially) leaky and tainted channels on the security of common authentication credentials, i.e., passwords and cookies.

5.6.1 Security analysis

In a typical web session, a website authenticates a user by checking her access credentials in the form of a username and a password. Upon their successful verification, the website stores in the user's browser a set of *session cookies*, which are automatically attached to the next requests sent to the website in order to authenticate them. There are quite a few well-known security threats in this common scenario [Cal+17] and vulnerable HTTPS implementations may severely compromise the security of web sessions. For example, if a user's password is disclosed to the attacker, the attacker will become able to start new sessions on the user's behalf and impersonate her at the website. Moreover, web session security requires both the confidentiality and the integrity of session cookies: lack of the former allows the attacker to hijack the user's session [Bug+15], while lack of the latter allows the attacker to force the user in the attacker's session [Zhe+15]. Though the latter threat is easily underestimated, it may have serious security consequences on many web applications: for instance, e-payment websites

may be targeted by such attacks to fool honest users into storing their credit card numbers in an attacker-controlled session.

Confidentiality of passwords

A critical requirement for the confidentiality of passwords is that they are only input on HTTPS pages and only sent over HTTPS channels. Modern web browsers indeed warn users when these security important requirements are not met [Sch17]. Unfortunately, vulnerable HTTPS implementations may make this security check insufficient: password confidentiality cannot be ensured when the password is sent over a leaky channel or entered into a compromisable web page where the attacker can get active scripting capabilities, thus becoming able to leak the password from the DOM.

Definition 5 (Low Confidentiality Password)**.** *A password has* low confidentiality *if and only if any of the following conditions holds:*

1. *the password is submitted over a leaky channel;*

2. *the page where the password is input is compromisable.*

Notice that partially leaky channels cannot be exploited to steal passwords, because the *secret repetition* assumption required by such side-channels is not satisfied by them.

Confidentiality of cookies

The confidentiality of cookies against network attackers can be enforced by means of the *Secure* attribute, because browsers ensure that Secure cookies are only sent on HTTPS channels and only made accessible to scripts running in HTTPS pages [Bar11]. However, this defense mechanism becomes useless when HTTPS does not provide the expected security guarantees: for example, even partially leaky channels may be sufficient to disclose the content of Secure cookies, since cookies are automatically attached by browsers and thus satisfy the *secret repetition* assumption required by attacks like POODLE-TLS. Moreover, compromisable pages can be exploited to steal Secure cookies by means of malicious scripts which exfiltrate them, unless these

cookies are also protected with the *HttpOnly* attribute, which prevents script accesses to them.

To make this intuition more precise, given a cookie c, we let $hosts(c)$ note the set of the hosts matching the domains which are entitled to access the content of c, as prescribed by RFC 6265 [Bar11]. Intuitively, c is attached to a request towards h if and only if $h \in hosts(c)$.

Definition 6 (Low Confidentiality Cookie)**.** *A cookie c set by the host h has* low confidentiality *if and only if any of the following conditions holds:*

1. *there exists a host $h' \in hosts(c)$ which allows for the establishment of partially leaky channels;*
2. *c does not have the HttpOnly attribute set and there exists a compromisable host $h' \in hosts(c)$.*

Notice that breaking the confidentiality of a single session cookie may not be enough to let the attacker hijack the sessions of legitimate users, because websites may use multiple cookies for authentication purposes [Cal+15]. However, if *all* the session cookies of a website have low confidentiality, we have definite evidence that there is room for session hijacking.

Integrity of cookies

Cookie integrity has notoriously been a major problem on the Web for many years, because cookies do not provide isolation by protocol, hence HTTP traffic can be abused to forge cookies which are indistinguishable from legitimate cookies set over HTTPS [Bar11]. Also, cookies can be set by potentially untrusted *related* domains, i.e., domains that share a common suffix which is not included in the Public Suffix List.[6] The recommended way to enforce cookie integrity against network attacks on the current Web is configuring HSTS so that all the hosts entitled to set cookies can only be contacted over HTTPS [Zhe+15]. An alternative approach is using *cookie prefixes*,[7] a recent addition to web browsers which can be used to prevent the setting of cookies over HTTP (when the *__Secure-* prefix appears in the cookie

6. `https://publicsuffix.org/`
7. `https://tools.ietf.org/html/draft-ietf-httpbis-rfc6265bis-02`

name) and, potentially, also from untrusted related domains (when the *__Host-* prefix appears in the cookie name, preventing cookie sharing between related domains). Unfortunately, these defenses might fail when HTTPS suffers from cryptographic flaws, because compromisable hosts would allow the attacker to break cookie integrity by corrupting HTTPS traffic; in particular, if the __Host- prefix is not used, any compromisable host on a related domain would be enough for the attack.

More precisely, given a host h, we let $related(h)$ note the set of the hosts whose domain is related to the domain of h. Technically, this implies that any host $h' \in related(h)$ can set a cookie c such that $h \in hosts(c)$, which means that c might be eventually received by h and harm its security. Notice that, although h' may not be able to directly overwrite host-only cookies set by h, it could still obtain the same effect by *cookie shadowing*, i.e., by setting domain cookies with the same name of host-only cookies so that the target website is fooled into accessing the former [Zhe+15]. Also, the domain cookies may be set before the host-only cookies are ever issued, which makes cookie shadowing attempts undetectable in general.

Definition 7 (Low Integrity Cookie)**.** *A cookie c set by the host h has* low integrity *if and only if any of the following conditions holds:*

1. *h is compromisable;*
2. *c does not have the __Host- prefix and there exists a compromisable host $h' \in related(h)$.*

5.6.2 Experimental results

We first isolated from the 10,000 crawled websites the 4,018 websites with a private area, i.e., supporting the establishment of authenticated sessions. This was assessed heuristically by checking any of the following two conditions:

1. the page includes a login form, i.e., a form with both a text/email field and a password field;
2. the page includes a single sign-on library from a list of popular identity providers.

	All cookies		
	Host-only (11,784)	Domain (31,998)	Total (43,782)
Confidentiality	1,469 (12.5%)	6,903 (21.6%)	8,372 (19.1%)
Integrity	2,093 (17.8%)	6,116 (19.1%)	8,209 (18.7%)

	Session cookies		
	Host-only (3,942)	Domain (7,818)	Total (11,760)
Confidentiality	425 (10.8%)	1,633 (20.1%)	2,058 (17.5%)
Integrity	694 (17.6%)	1,435 (18.3%)	2,129 (18.1%)

Table 5.3: Cookie confidentiality and integrity issues.

Out of the 4,018 websites with a private area, we found 404 cases where password confidentiality was not ensured (10.0%), either because the password was sent over a leaky channel or because the page with the login form was compromisable. Attacks against these pages would allow an attacker to impersonate legitimate users and start new sessions on their behalf.

We then turned our attention to the security analysis of cookies. The left portion of Table 5.6.1 reports the number of low confidentiality and low integrity cookies collected from the full set of 10,000 websites. In total, 19.1% of all cookies have low confidentiality, while 18.7% have low integrity, which suggests that the risks of cookie leakage and cookie tampering in the wild are far from remote. The most interesting observation is that ensuring confidentiality for domain cookies is much harder than for host-only cookies: 21.6% of the domain cookies have low confidentiality, while this percentage decreases to 12.5% for host-only cookies. The reason is that the attack surface for domain cookies is much larger, because it is enough to find one related domain which suffers from confidentiality issues to leak them; yet, 73.1% of the collected cookies are domain cookies. As to integrity, the difference between domain cookies and host-only cookies is almost negligible and the most concerning observation there is that only one of the 10,000 websites we crawled makes use of cookie prefixes to improve cookie integrity.

To better understand the impact of these numbers on web session security, we restricted our attention just to the session cookies set from

the 4,018 websites featuring a private area. Session cookies were identified using a heuristic proposed in previous work [Bug+15], which was shown to be fairly accurate in practice and nicely fits our large-scale investigation. The right portion of Table 5.6.1 presents the results of such analysis, which shows that the high-level picture does not change significantly when we focus just on session cookies. Moreover, we observed that 412 websites (10.2%) may leak all their session cookies due to cryptographic flaws, which may allow network attackers to impersonate legitimate users of these websites. It is worth noticing that, if all these cookies could be marked as HttpOnly without breaking the functionality of the websites, the number of websites vulnerable to this threat would reduce to 207 (5.1%). This shows that a complete deployment of the HttpOnly attribute would be quite effective, yet not sufficient to fully protect honest users against session hijacking, since session cookies could still be sent over partially leaky channels.

Finally, we found 543 websites (13.5%) whose session cookies all have low integrity, which may allow the attacker to force honest users into attacker-controlled sessions (cookie forcing). In all cases, the cookie integrity problems were due to the presence of a vulnerability in a related domain, but we also found 404 cases where also the base domain suffers from integrity flaws. The __Host- cookie prefix would be useful to improve session security in the 139 cases (25.6%) where the integrity vulnerabilities are confined to related domains, but unfortunately only one of the crawled websites (dropbox.com) uses cookie prefixes. Remarkably, we observe that 22 out of these 139 cases (15.8%) could safely introduce the __Host- prefix without compatibility problems, as none of their session cookies is a domain cookie.

5.6.3 Detected attacks

Since the numbers in the previous section may have been affected by the use of heuristics to detect private areas and session cookies, we report on a selected set of manual experiments to confirm the existence of credential stealing and session hijacking attacks on prominent websites in the wild. For ethical reasons, we did not tamper with websites to test concrete attacks. Rather, we carefully checked all the conditions

required to mount attacks against the targets and employed a local proxy to simulate the attack.

One notable example where password confidentiality is not ensured is Myspace. The login page and the endpoint where the password is sent are both served on myspace.com, that is directly vulnerable to ROBOT. Thus, an attacker could either sniff the password from a tainted channel or actively inject a script in the page to leak access credentials from the DOM.

Session hijacking has been identified as a realistic threat on the yandex.com web portal. In this case the main host itself is secure, but the presence of a partially leaky channel on api.developer.store.yandex.com makes possible for an attacker to disclose all domain cookies by forcing the victim's client to iterate requests against that specific host from an attacker's controlled origin. All cookies set by the website after logging in are domain cookies, including `Session_id` that is used to authenticate user sessions, proving the attack to be practical.

Finally, cookie forcing has been found on the Microsoft webmail live.com. Our large-scale assessment found that the host exchange.backcountry.com is vulnerable to ROBOT and presents a certificate valid also for outlook.live.com. Since the host of one of the related domains of live.com is compromisable, an attacker could mount a MITM to overwrite the cookies of a honest user, forcing her into the attacker's session.

5.7 Web tracking

In this section, we discuss how leaky and tainted channels can be abused to track navigation behaviours of web users and breach privacy at scale.

5.7.1 Security analysis

Online tracking is pervasive on the Web and has significant privacy implications [RKW12; EN16]. Third-party tracking is particularly dangerous for user privacy, because it allows trackers to reconstruct a cross-site navigation profile of online users at scale. In this form of

tracking, the tracker is embedded on external websites in a *third-party* position, i.e., using iframes, so that it is able to set a tracker-owned cookie containing a unique identifier in the user's browser. Every time the user accesses a website where the tracker is present, her browser will automatically send a request including the cookie to the tracker: since this request also includes the Referer header, which tracks the page from which the request was sent, the tracker becomes able to reconstruct the navigation profile of the user identified by the cookie.

Network attackers can easily disclose a lot of information about navigation patterns just because they are in control of the network. For instance, they can link a given IP address to all the domain names requested from it. However, this does not necessarily allow the attacker to build a navigation profile of the target user, e.g., because the same IP address is shared by multiple users (in case of NATs) or because the same user is assigned different IP addresses upon different connections. Still, it is known that network attackers may become able to build cross-site navigation profiles of users by monitoring the presence of tracking cookies in the HTTP traffic [Eng+15]. Here we discuss a similar attack, which exploits existing confidentiality issues in the HTTPS implementations of web trackers.

Assume the attacker wants to learn whether a user identified by the tracking cookie c has ever accessed the page p. If the page p includes sub-resources from a tracker-controlled host $h \in hosts(c)$ over a leaky channel, the attacker may be able to associate the value of c to the page p via the Referer header. However, even if p does not include anything from the tracker, the attacker can force such leaky content inclusion when p itself is compromisable, thus amplifying the privacy risks. This leads to the following definition.

Definition 8 (Profiling)**.** *A tracking cookie c allows* profiling *on the page p if and only if there exists a host $h \in hosts(c)$ which allows for the establishment of leaky channels and any of the following conditions holds:*

1. *p sends a request to h;*

2. *p is compromisable.*

Tracker	*Including Websites*
snap.licdn.com	126
l.betrad.com	100
hbopenbid.pubmatic.com	76
kraken.rambler.ru	66
ads.pubmatic.com	47
simage2.pubmatic.com	30
counter.rambler.ru	25
tag.1rx.io	20
fw-sync.nuggad.net	18
t.pubmatic.com	17

Table 5.4: Top trackers introducing privacy flaws.

5.7.2 Experimental results

We downloaded a list of 2,399 prominent tracking domains provided by Disconnect[8] and we checked for content inclusions from them in the 10,000 websites taken from Alexa. In particular, we focused on inclusions from any sub-domain of the trackers, because domain cookies could be used to perform tracking when including contents (of any type) from them. By doing this, we managed to identify a set of 4,226 tracker-controlled hosts which may potentially be abused to perform user profiling on the Alexa websites. We then analyzed these hosts, checking whether they allow the establishment of leaky channels, and it turned out that 82 (1.9%) of them suffer from this security issue.

We report in Table 5.4 the list of the most popular vulnerable tracker-controlled hosts, along with the number of websites from Alexa which included contents from them. These vulnerable hosts are controlled by different companies basing their business on web tracking and analytics. By checking against Cookiepedia,[9] we confirmed that at least four of these companies rely on the practice of setting long-lived domain cookies for third-party tracking: PubMatic, Rambler, RhythmOne and nugg.ad. To understand the privacy impli-

8. `https://github.com/disconnectme/disconnect-tracking-protection`
9. `https://cookiepedia.co.uk/`

cations of these security issues, we focused on the hosts controlled by PubMatic, which are the most numerous: attacking the vulnerable hosts of PubMatic would allow one to reconstruct navigation profiles over 142 websites which include contents from them. Moreover, by injecting references to these hosts in any of the 898 compromisable homepages from our dataset, this privacy attack could be further amplified to track navigation behaviors across 968 websites (9.7%).

5.8 Closing remarks

5.8.1 Related work

Novel attacks against TLS were often released with the analysis of their impact in the wild, by measuring the number of vulnerable servers in scans of the IPv4 address space or the most popular websites ranked by Alexa. This was true for RSA keys factorable by Batch GCD algorithm [Hen+12] and attacks like DROWN [Avi+16] or Logjam [Adr+15]. Small subgroup attacks against Diffie-Hellman were measured by Valenta et al. [Val+17]. Dorey et al. [DCE17] measured misconfigured DH key parameters that potentially contain backdoors. The prevalence of several attacks against the Elliptic Curve DH key establishment in TLS was measured by Valenta et al. [Val+18]. Some vulnerability measurements were revisited to track the progress of patching, such as Heartbleed [Dur+14] and the Batch GCD method [HFH16]. The SSL Pulse project [Qua18] releases monthly measurements on the prevalence of certain attacks and feature support. Novel variants of old vulnerabilities were discovered, such as in the ROBOT attack [BSY17], or for CBC oracles via the TLS-Attacker fuzzing tool [Som16b]. Summaries of known TLS vulnerabilities were published by Levillain et al. [Lev17; LGD15] and by the IETF [SHS15]. Lessons learned from attacks known before 2013 have been summarized by Meyer and Schwenk [MS14].

None of the papers above systematically discusses and quantifies web application security issues. However, the risks coming from the partial adoption of HTTP on HTTPS websites have been studied in several research papers. For instance, [Che+13] performed a large-scale analysis of the security risks of mixed content websites, [KB15] analyzed the state of the HSTS deployment and [SPK16] studied the

threats posed by the leakage of cookies over HTTP channels. There are also a few papers quantifying how much incorrect TLS implementations affect the security of the email infrastructure [Dur+15b; Hol+16].

The present chapter contributes to the increasingly popular research line on large-scale security evaluations of the Web [Goe+14]. Though several papers analyzed the security of the HTTPS certificate ecosystem [Dur+13; Hol+11; Van+16], we are not aware of any scientific publication which quantifies how much cryptographic weaknesses in TLS implementations may harm web application security. Other important aspects of web application security which have been investigated by previous large-scale measurements include the dangers of remote JavaScript inclusion [Nik+12], the prevalence of DOM-based XSS [LSJ13] and the state of the CSP adoption [CRB16; CRB18; Wei+16].

5.8.2 Ethics and limitations

Due to both legal and ethical reasons, our analysis of TLS vulnerabilities in the wild was limited to an unintrusive scan based on the use of publicly available tools. The exploitability of the discovered vulnerabilities was exclusively judged through a systematic analysis of the output of those tools, defined via an extensive account of the existing literature on attacks against TLS (summarized in the attack trees of Section 5.3). All the vulnerabilities we tested have been first published at major computer security conferences and/or received extensive coverage in the hacking community. They have all been shown to be exploitable in the wild, requiring a practically feasible amount of computational power. Since we did not run any active attack attempt, it is possible that the vulnerabilities reported in the present study are not actually exploitable in practice, e.g., due to the deployment of anomaly detection systems. That said, the real effectiveness of such kind of mitigations is hard to assess and fixing the vulnerabilities would be certainly preferable from a security perspective.

The set of the studied web application vulnerabilities is not intended to be exhaustive: it just gives evidence of significant security threats posed by vulnerable TLS implementations and allows for a systematic quantification of their practical relevance. The usage of

heuristics in a few parts of our experimental evaluation, e.g., for session cookie detection, may have introduced a bias in our quantitative assessment: better heuristics may make the analysis more precise, but they are likely not going to entail a significant change of the currently drawn picture, given the large scale of the experiments. We manually confirmed some of the security issues to provide further evidence of the effectiveness of our methodology. We also rechecked all the vulnerable sites explicitly mentioned in this chapter at the beginning of January 2019 and most of them have fixed the issues since our first scan. We have responsibly reported the discovered flaws to the sites that are still vulnerable and only one has answered dismissively with: "this case has no direct security impact and we will not take an immediate action or a fix". In fact, we did not find a strong interest in TLS-related issues even in vulnerability reward programs but the fact that many sites fixed the problems is promising in terms of awareness of the risks due to wrong HTTPS implementations.

5.8.3 Summary and perspective

Though the use of HTTPS is necessary for web application security, it is not a panacea, because flaws in the underlying TLS implementation may have a significant security impact at the application layer. We have computed a few disquieting numbers in our present evaluation: we summarize here the most relevant observations and present our perspective on the main findings.

Almost 10% of the homepages of the crawled websites is *compromisable*, i.e., a determined network attacker may get active scripting capabilities on them. For approximately 25% of the compromisable pages, this security problem can be fixed just by revising the cryptographic implementation of their host. Unfortunately, the security of the other 75% pages is downgraded by the inclusion of external scripts retrieved over tainted channels: this makes it hard for web developers to get a realistic picture of the cryptographic robustness of their web applications and fix potential issues. Since we only crawled homepages, our findings under-approximate the real situation, as other webpages might include more insecure content. SRI is a potentially effective defense mechanism for these cases, but its adoption is minuscule and sub-optimal: approximately, just 3% of the pages are

using SRI and none of the attacks we found is actually stopped by the current deployment.

For what concerns web session security, we found room for session hijacking attacks by cookie stealing in around 10% of the crawled websites, while more than 13% of the websites were found vulnerable to cookie forcing. The most concerning aspect of cookie security is the impact of related domains: even a single security issue on a related-domain host may completely undermine session security, because related-domain hosts may break both the confidentiality and the integrity of session cookies. Room for password theft was also found in 10% of the login pages.

Finally, cryptographic weaknesses in the TLS implementations of web trackers may pose major threats to user privacy at scale. In our experimental analysis, we discovered some prominent trackers inadvertently introducing this security problem on a significant amount of websites. The most disquieting aspect here is that just a single vulnerable tracker may significantly harm user privacy at scale, as long as it is popular enough to be included on many different websites: for instance, one problem we found allows for user profiling on 142 websites, which can be further increased to 968 websites by running a more powerful variant of the attack.

We expect this bleak picture to improve after both browsers and servers provide a better support for TLS 1.3. Major browser vendors already announced that they will deprecate TLS 1.0 and 1.1 in 2020 [Ben18]. However, backward compatibility and slow adoption are always a major hindrance for web security improvements, so we expect old TLS versions to stick around for at least a few years. The present study acts as a cautionary tale of the threats they pose: we plan to supply the toolchain developed for our study as a web application to support developers who are interested in mitigating these threats.

5.9 Additional results

5.9.1 Notable out of scope attacks against TLS

Several vulnerabilities of TLS are not exploitable in the wild, based on recent measurements or due to the configuration of modern clients.

Diffie-Hellman key establishment attacks (MITM attacks)

Static DH key exchange susceptible to small subgroup attacks [Val+17] is not supported by modern browsers and support for vulnerable static ECDH key exchange was removed in browsers we target. Furthermore, some browsers already deprecated DHE [DCE17] and more should follow. Possibly backdoored DH groups were observed in the wild [DCE17]. It is not possible to intercept the connection without the knowledge of the backdoor, hence only the attacker that generated the backdoored parameters could mount MITM attacks. The Logjam attack [Adr+15] forces the server to choose a small 512-bit DH group, however modern browsers enforce minimal group size, where the discrete logarithm problem is infeasible.

A recent paper [Val+18] measured the prevalence and feasibility of several attacks on ECDH (static and ephemeral) key establishment. Many servers fail to check parameters and many reuse ephemeral keys [SDH16], no server was found that would do both. Their further findings indicate that several other proposed attacks (such as CurveSwap) are infeasible in TLS.

State machine bugs (up to MITM)

The state machines of TLS are complicated and not explicitly stated in the standards. Their implementations are a common source of bugs. The Early CCS attack found by [Kik14] allowed a MITM attack. Due to a bug in OpenSSL, running the Change Cipher Spec Protocol early, both the server and the client used a zero-length master key. While the bug is still found on some servers [Qua18], browsers have been patched. FREAK, another client-side bug [Beu+15] allowed the attacker to downgrade the client to `RSA_EXPORT` (easily factorable 512-bit keys), even when the client did not offer such ciphersuite. Searching for new state machine bugs was out of our scope and is the focus of systematic studies of state machine implementations [Beu+15; RP15].

Private key leakage (MITM)

Private RSA keys generated with insufficient entropy can lead to servers sharing primes in their keys, allowing such RSA keys to be factored by a simple greatest common divisor (GCD) computation. Batch

GCD, an efficient version of the algorithm that can handle millions of moduli, revealed that such keys were widespread [Hen+12; HFH16], likely due to consumer devices that generate their keys shortly after boot, before entropy is collected. The bugs are not prevalent on commercial servers from the Alexa list.

DSA and ECDSA private keys can be recovered if the same secret nonce is used more than once [Nat13], yet it happens with negligible probability. Even biased nonces can be used to reveal the private key, if enough signatures with a small number of known nonce bits are known [Shp03]. However, testing for such side-channels is infeasible. Remote time side-channel attacks were demonstrated [BT11], yet the bugs were known beforehand. Timing attacks often rely on observing cache access [RPS18] that cannot be performed from a MITM position.

Certificate validation bugs (MITM)

Some non-browser clients were shown to have flawed certificate validation [Geo+12], accepting invalid certificates. We assume correct certificate validation in modern browsers and users following browser warnings. Certificate validation bugs in software and hardware that intercepts TLS connections [CM16; Dur+17; WMY18] are also out of scope of our analysis.

Transcript collision attacks (MITM)

We leave out transcript collision attacks [BL16a] since the performance of the algorithms for finding (chosen prefix) collision in the hash functions is not yet practical enough.

Further CBC-mode attacks (partial secret leakage)

Attacks based on timing side-channels like Lucky13 [AP13] were infeasible for us to assess over the Internet. The original POODLE attack [MDK14] cannot be applied, since browsers disabled SSLv3 support. Browsers that fix bugs, such as an SOP-bypass, or implement the 1/n-1 split will resist BEAST [DR11]. We leave for future work the attacks that enable partially leaky channels from server to client, like

BREACH [PHG13], that requires specific conditions at the server's application layer to be exploited.

Weak ciphers (partial secret leakage)

Authentication tokens and cookies could be disclosed due to collisions in CBC mode of a 64-bit block cipher, such as Triple-DES (3DES), via the Sweet32 attack [BL16b]. Due to the birthday paradox, a ciphertext collision between a block that encrypts a known plaintext and a block that encrypts the cookie is expected with high probability after the client sends about 2^{32} messages. Modern browsers may support 3DES as a fallback in case that the preferred AES cipher (with at least 128-bit blocks) is not supported by the server. An effective mitigation is to disable 3DES support or enforce a conservative bound for the amount of data encrypted under one key. We assumed such limit in browsers and therefore removed the attack from scope.

It is possible to extract short secrets using a statistical attack against the biased key stream of the RC4 stream cipher [GPM15]. Although the current state of the art attack still requires a large number of secret repetitions, IETF deprecated RC4 use in TLS [Pop15] and major browsers disabled RC4 support.

Compression oracles (partial secret leakage)

A side-channel based on compression was described by Kelsey [Kel02]. If the attacker injects into the plaintext a copy of the secret, the compression should reduce the size of the ciphertext, when compared to injecting random plaintext of the same size. The attacker could observe the size of the ciphertext (CRIME attack [RD12]) or the time of the transmission (TIME attack [BS13]) to build an oracle for verifying guesses of the secret. The attacks require secret repetition and partial control over the plaintext. Modern clients disable compression of TLS records, and so does the majority of the servers [Qua18].

Renegotiation and Triple Handshake (integrity)

We consider the Renegotiation attack [RD09] and the Triple Handshake attack [Bha+14] as out of scope. The main idea of the attacks is that the

```
GOAL Bleichenbacher's oracle on the server
| 1 The response to any of these client key
    exchanges differs:
  | 1 Correct padding:
      00 02 <random> 00 <TLS version> <PMS>
  | 2 Wrong first two bytes:
      41 17 <random> 00 <TLS version> <PMS>
  | 3 A 0x00 byte in a wrong position:
      00 02 <random> 11 <PMS> 00 11
  | 4 Missing 0x00 byte in the middle:
      00 02 <random> 11 11 11 <PMS>
  | 5 Wrong version number oracle [KPR03]:
      00 02 <random> 00 02 02 <PMS>
```

Figure 5.5: A simplified test for general Bleichenbacher's oracle from testssl.sh.

messages sent by the client are "spliced" into ongoing communication between the attacker and the server, and the server assumes continuity before and after renegotiation, despite TLS not giving such guarantee. We do not consider Client Authentication and do not test application layer authentication for such behavior.

5.9.2 More detailed attack trees

Tests performed by security tools can be also described as attack trees. To illustrate the specific conditions of some attacks, we present an abstraction of the tests for Bleichenbacher's oracle in Figure 5.5 and its Strong variant in Figure 5.6, General and Special DROWN attack in Figure 5.7 and Figure 5.8, respectively, and the conditions for POODLE-TLS in Figure 5.9 and for a specific CBC padding oracle in Figure 5.10.

Some leaf conditions in the trees are represented by sub-trees. We list some of them explicitly, namely the requirements for an attacker to mount a protocol version downgrade attack (Figure 5.11), the conditions indicating the presence of an RSA decryption oracle (Figure 5.12 and 5.13), and the tree for fast RSA signature oracle (Figure 5.14). Other leaf conditions are more intuitive or they are mapped to the outputs of the attack vulnerability testing tools, testssl.sh, TLS-Attacker [Som16b], and the DROWN detection plugin for nmap.

```
GOAL Strong Bleichenbacher's oracle on the server
& 1 Bleichenbacher's oracle on the server (Figure 5.5)
& 2 The client key exchange messages 2, 3, and 4
    invoked at least 2 different server responses
```

Figure 5.6: A simplified test for Strong Bleichenbacher's oracle from testssl.sh.

```
GOAL Server is vulnerable to General DROWN
| 1 Server supports a vulnerable SSLv2 ciphersuite
    (using DES or a cipher with 40-bit keys)
  | 1 Server offers such ciphersuite (CVE-2016-0800)
  | 2 Server accepts such ciphersuite without
      advertising its support (CVE-2015-3197)
```

Figure 5.7: The test for General DROWN according to the detection script (the test is repeated for different application protocols).

```
GOAL Server is vulnerable to Special DROWN
& 1 Server supports SSLv2
& 2 Server has the "extra clear" oracle (it allows
    clear_key_data bytes for non-export ciphers)
```

Figure 5.8: The test for Special DROWN according to the detection script.

```
GOAL POODLE-TLS padding oracle on the server
| 1 Server does not respond with a Fatal Alert to
    a message with an error on the first byte of the
    padding (the rest of the padding is correct)
```

Figure 5.9: The test for a POODLE-TLS padding oracle as seen in TLS-Attacker.

```
GOAL CBC padding oracle CVE-2016-2107 on the server
| 1 Server issues a RECORD_OVERFLOW alert
    as a response to a specially crafted message
```

Figure 5.10: The test for a CBC padding oracle due to an OpenSSL bug in AES-NI code (CVE-2016-2107) as seen in TLS-Attacker (simplified).

```
GOAL Downgrade to a specific lower protocol version <V>
& 1 At least one of the peers lacks version downgrade mitigation
  | 1 Client does not support RFC 7507 TLS_FALLBACK_SCSV
      (i.e., the Client does not append the ciphersuite to a
      ClientHello with other than the highest supported TLS version)
  | 2 Server does not support RFC 7507 TLS_FALLBACK_SCSV
      (i.e., the Server does not check for the presence
      of the ciphersuite  in the ClientHello)
& 2 Client and Server support a specific lower protocol version <V>
    (with some interesting property, e.g., with preferred CBC mode
    of symmetric encryption, or only supporting RSA key exchange)
  & 1 Server supports the lower protocol version <V>
  & 2 Client supports the lower protocol version <V> (e.g., modern
      web browsers support TLS 1.0 to 1.3, but not SSLv2 and SSLv3)
```

Figure 5.11: Attack sub-tree for protocol version downgrade.

```
GOAL RSA decryption oracle is available
| 1 Oracle allows feasible decryption
  | 1 Strong Bleichenbacher's oracle on the server (Figure 5.6)
  | 2 General DROWN
    & 1 Server is vulnerable to General DROWN (Figure 5.7)
    & 2 Attacker can capture a key exchange in the required format
        (1 in 900) (assumption)
| 2 Fast RSA decryption oracle (Figure 5.13)
```

Figure 5.12: Attack sub-tree for an RSA decryption oracle (that allows a decryption of key exchange messages).

```
GOAL Fast RSA decryption oracle
| 1 Strong Bleichenbacher's PKCS #1 v1.5 oracle and high performance
  & 1 Strong Bleichenbacher's oracle on the server (Figure 5.6)
  & 2 Attacker can decrypt before the handshake finishes
      (assumption about the performance of the Server and Attacker
      to handle many parallel connections)
| 2 Special DROWN
  & 1 Server is vulnerable to Special DROWN (Figure 5.8)
  & 2 Attacker can capture a key exchange in the required format
      (1 in 260) (assumption)
```

Figure 5.13: Attack sub-tree for a fast RSA decryption oracle (that allows an online decryption).

```
GOAL Fast RSA signature oracle
| 1 Strong Bleichenbacher's PKCS #1 v1.5 oracle and high performance
  & 1 Strong Bleichenbacher's oracle on the server (Figure 5.6)
  & 2 Attacker can forge the signature before the handshake finishes
      (assumption about the performance of the Server and Attacker
      to handle many parallel connections)
```

Figure 5.14: Attack sub-tree for a fast RSA signature oracle (that allows an online decryption or signature forgery).

6 Conclusions

This thesis advances the security of real-world systems and cryptographic implementations and illustrates several interesting points. We showed that seemingly small decisions in the design of an RSA key generation algorithm could have surprising consequences. Even correct but distinct approaches reveal some information about the system. We saw a deviation from the norm cause devastating consequences for the security of an algorithm. Finally, when considering the security of a complicated system, such as Internet servers running TLS protocol implementations, the scope of the study can change the results dramatically. While many measurements focus on popular websites, their supporting infrastructure tends to receive less attention, while potentially hiding more issues.

The Internet is exceptionally complicated. We helped the community understand some of its security aspects a little better. Our approach to measuring the popularity of cryptographic libraries leverages slight biases in the distributions of RSA keys generated by the libraries. We saw interesting trends that indicate that many servers rely on a few popular libraries, with OpenSSL in the lead. Different application domains have varying needs, and understanding them is the first step towards improving their security. To improve our analysis technique in the future, we encourage the use of data mining to obtain more value from the data than summaries and statistics. It would also be valuable to extend the measurement method to other cryptographic algorithms. To developers of cryptographic libraries, we recommend following a standardized RSA key generation approach to minimize leaking the details of their implementation and to help prevent design mistakes.

In our work on the security of the TLS protocol, we focused on the outliers that have exploitable security weaknesses. By studying the relatively few broken implementations in the interconnected context of the Web, we saw that their weaknesses get amplified by systems that rely on them. Many papers that measure the prevalence of cryptographic attacks only evaluate primary targets and neglect to consider the surrounding infrastructure. We encourage security researchers to expand their analyses and practitioners to seek safe configurations.

An example of a potentially unsafe server configuration was the case of reusing RSA keys or TLS certificates amongst servers with different levels of attention to timely security updates. We also saw the importance of keeping the vulnerability tests lightweight, since the scope quickly expands, when connections begin to be explored.

In both of the wide-scale measurements, it was crucial to present the results in an accessible way. For the measurement of TLS attacks, we described the conditions and effects of the attacks in the form of attacks trees. The trees break a complicated topic into smaller parts that are easier to understand and help highlight the important outcomes at the top of the tree. For the popularity measurements, we transformed a large dataset into concise summaries and graphs. Similar methods could be applied to other studies as an avenue for future work.

Studying an outlier in the area of RSA key generation lead us to the discovery of a severe flaw affecting a cryptographic library used in many domains based on secure chips. The research resulted in a practical factorization method, but it also brought to light other interesting questions. A disadvantage of a proprietary design was demonstrated. The discovery of the issue was delayed for many years, possibly due to the secret specification and implementation. A weakness of the certification process was illustrated, where the library received security certifications as a whole, while an important part was not considered for evaluation.

In future work, we aim to study more potential outliers and to look for other features and transformations that could reveal such oddities. We plan to evaluate both open and closed systems since neither the open-source nature of the code nor the process of private certifications can guarantee security by itself.

The most critical point that connects all our research questions is that providing developers and users with many choices might be harmful to security. There is no consensus on what RSA key generation method is the most robust. Details are often left out of specifications, such as low-level implementation decisions when generating RSA keys. The developers were overwhelmed by a repeatedly expanding list of countermeasures needed to make the use of RSA in TLS secure from padding oracle attacks.

In contrast, recent publications and standards using elliptic curve cryptography (ECC) are more authoritative. The recommended curves

are limited to robust choices, and the algorithm descriptions are more precise. That is not to say that ECC is without pitfalls, but the community has taken steps to limit both design and implementation mistakes. Decisions were lifted from the developers, and more secure choices were made for the standard applications of ECC. It might be a good idea to follow the example of TLS version 1.3 and to deprecate the use of RSA in favor of ECC. That might help escape the legacy of implementations that do not stand up to the challenge.

Bibliography

[Adr+15] D. Adrian, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, P. Zimmermann, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, and E. Thomé. "Imperfect Forward Secrecy". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security – CCS '15*. ACM Press, 2015. DOI: 10.1145/2810103.2813707.

[Akh+16] D. Akhawe, F. Braun, F. Marier, and J. Weinberge. *W3C Recommendation: Subresource Integrity*. 2016. URL: https://www.w3.org/TR/SRI/ (retrieved 2019-09-24).

[Alb+16] M. R. Albrecht, J. P. Degabriele, T. B. Hansen, and K. G. Paterson. "A Surfeit of SSH Cipher Suites". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. ACM, 2016, pp. 1480–1491. ISBN: 978-1-4503-4139-4.

[Alb+18] M. R. Albrecht, J. Massimo, K. G. Paterson, and J. Somorovsky. "Prime and Prejudice". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*. ACM Press, 2018. DOI: 10.1145/3243734.3243787. URL: https://doi.org/10.1145%2F3243734.3243787.

[AP13] N. J. AlFardan and K. G. Paterson. "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols". In: *2013 IEEE Symposium on Security and Privacy*. IEEE, May 2013. DOI: 10.1109/sp.2013.42.

[AlF+13] N. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. "On the Security of RC4 in TLS". In: *Proceedings of the 22nd USENIX Security Symposium (USENIX Security 13)*. USENIX, 2013, pp. 305–320. ISBN: 978-1-931971-03-4. URL: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/alFardan.

[Avi+16] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Hal-

derman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. "DROWN: Breaking TLS Using SSLv2". In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 2016, pp. 689–706. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram.

[AZ04] Axalto and Zetes. *Public user specification BelPic application v2.0*. 2004. URL: http://www.foo.be/eID/opensc-belgium/BEID-CardSpecs-v2.0.0.pdf (retrieved 2019-09-24).

[Bar+16] M. Barbulescu, A. Stratulat, V. Traista-Popescu, and E. Simion. "RSA Weak Public Keys Available on the Internet". In: *Innovative Security Solutions for Information Technology and Communications (SECITC 2016)*. Springer International Publishing, 2016, pp. 92–102. DOI: 10.1007/978-3-319-47238-6_6.

[Bar+12] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J.-K. Tsay. "Efficient Padding Oracle Attacks on Cryptographic Hardware". In: *Advances in Cryptology – CRYPTO 2012*. Springer Berlin Heidelberg, 2012, pp. 608–625. DOI: 10.1007/978-3-642-32009-5_36.

[Bar11] A. Barth. *RFC 6265: HTTP State Management Mechanism*. 2011. URL: http://tools.ietf.org/html/rfc6265 (retrieved 2019-09-24).

[BS13] T. Be'ery and A. Shulman. *A Perfect CRIME? Only TIME Will Tell*. 2013. URL: https://media.blackhat.com/eu-13/briefings/Beery/bh-eu-13-a-perfect-crime-beery-wp.pdf (retrieved 2019-09-24).

[Bel08] L. Bello. *DSA-1571-1 openssl – predictable random number generator*. 2008. URL: https://www.debian.org/security/2008/dsa-1571 (retrieved 2019-09-24).

[Ben18] D. Benjamin. *Modernizing Transport Security*. 2018. URL: https://security.googleblog.com/2018/10/modernizing-transport-security.html (retrieved 2019-09-24).

[Ber67] E. R. Berlekamp. "Factoring Polynomials Over Finite Fields". In: *Bell System Technical Journal* 46.8 (1967), pp. 1853–1859. ISSN: 1538-7305.

[Ber05] D. J. Bernstein. *Cache-timing attacks on AES*. 2005. URL: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf (retrieved 2019-09-24).

[Ber+13] D. J. Bernstein, Y.-A. Chang, C.-M. Cheng, L.-P. Chou, N. Heninger, T. Lange, and N. van Someren. "Factoring RSA Keys from Certified Smart Cards: Coppersmith in the Wild". In: *Advances in Cryptology - ASIACRYPT 2013*. Springer-Verlag, 2013, pp. 341–360. ISBN: 978-3-642-42045-0.

[BHL12] D. J. Bernstein, N. Heninger, and T. Lange. *Batch gcd*. 2012. URL: http://facthacks.cr.yp.to/batchgcd.html (retrieved 2019-09-24).

[Beu+15] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. "A Messy State of the Union: Taming the Composite State Machines of TLS". In: *2015 IEEE Symposium on Security and Privacy*. IEEE, May 2015. DOI: 10.1109/sp.2015.39.

[Bha+14] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub. "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS". In: *2014 IEEE Symposium on Security and Privacy*. IEEE, May 2014. DOI: 10.1109/sp.2014.14.

[BL16a] K. Bhargavan and G. Leurent. "Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH". In: *Proceedings of the 2016 Network and Distributed System Security Symposium*. Internet Society, 2016. DOI: 10.14722/ndss.2016.23418.

[BL16b] K. Bhargavan and G. Leurent. "On the Practical (In-)Security of 64-bit Block Ciphers". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*. ACM Press, 2016. DOI: 10.1145/2976749.2978423.

[Ble98] D. Bleichenbacher. "Chosen ciphertext attacks against protocols based on the RSA encryption standard

PKCS#1". In: *Advances in Cryptology – CRYPTO '98*. Springer Berlin Heidelberg, 1998, pp. 1–12. DOI: 10.1007/bfb0055716.

[BM06] D. Bleichenbacher and A. May. "New Attacks on RSA with Small Secret CRT-Exponents". In: *Public Key Cryptography - PKC 2006*. Springer-Verlag, 2006, pp. 1–13.

[BM03] J. Blömer and A. May. "New Partial Key Exposure Attacks on RSA". In: *Advances in Cryptology – CRYPTO 2003*. Springer-Verlag, 2003, pp. 27–43.

[BSY17] H. Böck, J. Somorovsky, and C. Young. *Return Of Bleichenbacher's Oracle Threat (ROBOT)*. 2017. URL: https://eprint.iacr.org/2017/1189 (retrieved 2019-09-24).

[BD99] D. Boneh and G. Durfee. "Cryptanalysis of RSA with Private Key d Less than $N^{0.292}$". In: *Advances in Cryptology — EUROCRYPT '99*. Springer-Verlag, 1999, pp. 1–11.

[BF97] D. Boneh and M. Franklin. "Efficient generation of shared RSA keys". In: *Advances in Cryptology-CRYPTO'97*. Springer. 1997, p. 425.

[Bor+09] M. Bortolozzo, G. Marchetto, R. Focardi, and G. Steel. "Secure your PKCS#11 token against API attacks!" In: *3rd International Workshop on Analysis of Security APIs (ASA-3)*. July 2009.

[BDL93] J. Brandt, I. Damgård, and P. Landrock. "Speeding up prime number generation". In: *Advances in Cryptology — ASIACRYPT '91*. Springer-Verlag, 1993, pp. 440–449.

[Bre80] R. P. Brent. "An improved Monte Carlo factorization algorithm". In: *BIT Numerical Mathematics* 20.2 (1980), pp. 176–184.

[Bro05] D. R. L. Brown. *A Weak-Randomizer Attack on RSA-OAEP with e = 3*. 2005. URL: http://eprint.iacr.org/2005/189 (retrieved 2019-09-24).

[BT11] B. B. Brumley and N. Tuveri. "Remote Timing Attacks Are Still Practical". In: *Computer Security – ESORICS 2011*. Springer Berlin Heidelberg, 2011, pp. 355–371. DOI: 10.1007/978-3-642-23822-2_20.

[BB05] D. Brumley and D. Boneh. "Remote timing attacks are practical". In: *Computer Networks* 48.5 (Aug. 2005), pp. 701–716. DOI: 10.1016/j.comnet.2005.01.010.

[BSI15] BSI. "Certification Report, BSI-DSZ-CC-0782-V2-2015, Infineon Security Controller M7892 B11 with optional RSA2048/4096 v1.02.013, EC v1.02.013, SHA-2 v1.01 and Toolbox v1.02.013 libraries and with specific IC dedicated software (firmware), v1.0," in: 2015. URL: https://www.commoncriteriaportal.org/files/epfiles/0782V2a_pdf.pdf (retrieved 2019-09-24).

[Bug+15] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan. "CookiExt: Patching the browser against session hijacking attacks". In: *Journal of Computer Security* 23.4 (2015), pp. 509–537.

[BE10] J. Burns and T. EFF. *The EFF SSL Observatory*. 2010. URL: https://www.eff.org/observatory (retrieved 2019-09-24).

[Cal+07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. *RFC 4880: OpenPGP Message Format*. 2007. URL: https://tools.ietf.org/html/rfc4880 (retrieved 2019-09-24).

[Cal+19] S. Calzavara, R. Focardi, M. Nemec, A. Rabitti, and M. Squarcina. "Postcards from the Post-HTTP World: Amplification of HTTPS Vulnerabilities in the Web Ecosystem". In: *IEEE S&P 2019*. 2019.

[Cal+17] S. Calzavara, R. Focardi, M. Squarcina, and M. Tempesta. "Surviving the Web: A Journey into Web Session Security". In: *ACM Comput. Surv.* 50.1 (2017), 13:1–13:34.

[CRB16] S. Calzavara, A. Rabitti, and M. Bugliesi. "Content Security Problems?: Evaluating the Effectiveness of Content Security Policy in the Wild". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 1365–1375.

[CRB18] S. Calzavara, A. Rabitti, and M. Bugliesi. "Semantics-Based Analysis of Content Security Policy Deployment". In: *TWEB* 12.2 (2018), 10:1–10:36.

[Cal+15] S. Calzavara, G. Tolomei, A. Casini, M. Bugliesi, and S. Orlando. "A Supervised Learning Approach to Protect Client Authentication on the Web". In: *TWEB* 9.3 (2015), 15:1–15:30.

[CZ81] D. G. Cantor and H. Zassenhaus. "A New Algorithm for Factoring Polynomials Over Finite Fields". In: *Mathematics of Computation* 36.154 (1981), pp. 587–592. ISSN: 00255718, 10886842.

[Can+03] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux. "Password Interception in a SSL/TLS Channel". In: *Advances in Cryptology – CRYPTO 2003*. Springer Berlin Heidelberg, 2003, pp. 583–599. DOI: 10.1007/978-3-540-45146-4_34.

[CM16] X. de Carné de Carnavalet and M. Mannan. "Killed by Proxy: Analyzing Client-end TLS Interception Software". In: *Proceedings 2016 Network and Distributed System Security Symposium*. Internet Society, 2016. DOI: 10.14722/ndss.2016.23374.

[Cen15a] Censys. *TLS Alexa Top 1 Million Scan*. 2015. URL: https://censys.io/data/443-https-tls-alexa_top1mil (retrieved 2019-09-24).

[Cen15b] Censys. *TLS Full IPv4 443 Scan*. 2015. URL: https://censys.io/data/443-https-tls-full_ipv4/historical (retrieved 2017-09-20).

[Che+13] P. Chen, N. Nikiforakis, C. Huygens, and L. Desmet. "A Dangerous Mix: Large-Scale Analysis of Mixed-Content Websites". In: *Information Security, 16th International Conference, ISC 2013, Proceedings*. 2013, pp. 354–363.

[Chu+16] T. Chung, Y. Liu, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. "Measuring and Applying Invalid SSL Certificates: The Silent Majority". In: *Proceedings of the 2016 ACM on Internet Measurement Conference – IMC '16*. ACM Press, 2016. DOI: 10.1145/2987443.2987454.

[CO13] J. Clark and P. C. van Oorschot. "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements". In: *IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 511–525.

[Col15] Y. Collet. *LZ4 Extremely Fast Compression algorithm*. 2015. URL: http://www.lz4.org/ (retrieved 2019-09-24).

[Cop96a] D. Coppersmith. "Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known". In: *Advances in Cryptology – EUROCRYPT '96*. Springer Berlin Heidelberg, 1996, pp. 178–189. DOI: 10.1007/3-540-68339-9_16.

[Cop96b] D. Coppersmith. "Finding a Small Root of a Univariate Modular Equation". In: *Advances in Cryptology — EUROCRYPT '96*. Springer-Verlag, 1996, pp. 155–165.

[Cop97] D. Coppersmith. "Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities". In: *Journal of Cryptology* 10.4 (1997), pp. 233–260. ISSN: 1432-1378.

[Cop+96] D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. "Low-Exponent RSA with Related Messages". In: Springer-Verlag, 1996, pp. 1–9.

[CDN05] R. Cramer, I. Damgård, and J. B. Nielsen. "Multiparty computation, an introduction". In: *Contemporary cryptology* (2005), pp. 41–87.

[CRo17] CRoCS MU. *JCAlgTest: JavaCard Algorithm Test*. 2017. URL: http://jcalgtest.org/ (retrieved 2019-09-24).

[DA99] T. Dierks and C. Allen. *RFC 2246: The TLS Protocol Version 1.0*. 1999. URL: https://tools.ietf.org/html/rfc2246 (retrieved 2019-09-24).

[DR06] T. Dierks and E. Rescorla. *RFC 4346: The Transport Layer Security (TLS) Protocol Version 1.1*. 2006. URL: https://tools.ietf.org/html/rfc4346 (retrieved 2019-09-24).

[DR08] T. Dierks and E. Rescorla. *RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2*. 2008. URL: https://tools.ietf.org/html/rfc5246 (retrieved 2019-09-24).

[DCE17] K. Dorey, N. Chang-Fong, and A. Essex. "Indiscreet Logs: Diffie-Hellman Backdoors in TLS". In: *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017. DOI: 10.14722/ndss.2017.23006.

[DR11] T. Duong and J. Rizzo. *Here Come The XOR Ninjas*. 2011. URL: https://bug665814.bugzilla.mozilla.org/attachment.cgi?id=540839 (retrieved 2019-09-24).

[Dur+15a] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. "A search engine backed by Internet-wide scanning". In: *Proceedings of the 22nd ACM SIGSAC Con-*

ference on Computer and Communications Security. ACM. 2015, pp. 542–553.

[DBH14] Z. Durumeric, M. Bailey, and J. A. Halderman. "An Internet-Wide View of Internet-Wide Scanning." In: *Proceeding of USENIX Security Symposium*. 2014, pp. 65–78.

[Dur+15b] Z. Durumeric, J. A. Halderman, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, and M. Bailey. "Neither Snow Nor Rain Nor MITM..." In: *Proceedings of the 2015 ACM Conference on Internet Measurement Conference - IMC '15*. ACM Press, 2015. DOI: 10.1145/2815675.2815695.

[Dur+13] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. "Analysis of the HTTPS certificate ecosystem". In: *Proceedings of the 2013 Internet Measurement Conference, IMC 2013*. 2013, pp. 291–304.

[Dur+17] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson. "The Security Impact of HTTPS Interception". In: *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017. DOI: 10.14722/ndss.2017.23456.

[Dur+14] Z. Durumeric, M. Payer, V. Paxson, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, and J. Beekman. "The Matter of Heartbleed". In: *Proceedings of the 2014 Conference on Internet Measurement Conference – IMC '14*. ACM Press, 2014. DOI: 10.1145/2663716.2663755.

[EMV11] EMVCo. "EMV Integrated Circuit Card Specifications for Payment Systems". In: 2011. URL: https://www.emvco.com/document-search/ (retrieved 2019-09-24).

[EMV12] EMVCo. *EMVCo Product Approval, ICCN0163, Master component: M7892 A22/B11, 11 Jan 2012*. 2012. URL: https://2426-9805.el-alt.com/loa_se/EMVCo_ICCN0163_R_02_2017.pdf (retrieved 2019-09-24).

[EMV13] EMVCo. *EMVCo Product Approval, ICCN0200, Master component: M7893 B11, 20 Dec 2013*. 2013. URL: https://2426-

9805.el-alt.com/loa_se/EMVCo_ICCN0200_R_02_2017.pdf (retrieved 2019-09-24).

[EN16] S. Englehardt and A. Narayanan. "Online Tracking: A 1-million-site Measurement and Analysis". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016.

[Eng+15] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten. "Cookies That Give You Away: The Surveillance Implications of Web Tracking". In: *Proceedings of the 24th International Conference on World Wide Web*. 2015, pp. 289–299.

[Fel+17] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz. "Measuring HTTPS Adoption on the Web". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1323–1338. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/felt.

[Gar95] S. Garfinkel. *PGP: pretty good privacy*. ISBN 978-1-56592-098-9. O'Reilly Media, Inc., 1995.

[GPM15] C. Garman, K. G. Paterson, and T. V. der Merwe. "Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS". In: *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015, pp. 113–128. ISBN: 978-1-931971-232. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/garman.

[GB17] I. Genibel and C. Berg. *Debian quality assurance: Popularity contest statistics*. 2017. URL: https://qa.debian.org/popcon.php (retrieved 2019-09-24).

[Gen+99] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Secure distributed key generation for discrete-log based cryptosystems". In: *Eurocrypt*. Vol. 99. Springer. 1999, pp. 295–310.

[Geo+12] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. "The most dangerous code in the world: validating SSL certificates in non-browser software". In: *Proceedings of the 2012 ACM conference on Com-*

puter and communications security – CCS '12. ACM Press, 2012. DOI: 10.1145/2382196.2382204.

[Gil99] N. Gilboa. "Two party RSA key generation". In: *Annual International Cryptology Conference*. Springer. 1999, pp. 116–129.

[Glo06] GlobalPlatform. *Card Specification Version 2.2*. 2006. URL: https://globalplatform.org/specs-library/ (retrieved 2019-09-24).

[Goe+14] T. van Goethem, P. Chen, N. Nikiforakis, L. Desmet, and W. Joosen. "Large-Scale Security Analysis of the Web: Challenges and Findings". In: *Trust and Trustworthy Computing - 7th International Conference, TRUST 2014. Proceedings*. 2014, pp. 110–126.

[Goo17a] D. Goodin. *Flaw crippling millions of crypto keys is worse than first disclosed*. Ars Technica. 2017. URL: https://arstechnica.com/information-technology/2017/11/flaw-crippling-millions-of-crypto-keys-is-worse-than-first-disclosed/ (retrieved 2019-09-24).

[Goo17b] D. Goodin. *Millions of high-security crypto keys crippled by newly discovered flaw*. Ars Technica. 2017. URL: https://arstechnica.com/information-technology/2017/10/crypto-failure-cripples-millions-of-high-security-keys-750k-estonian-ids/ (retrieved 2019-09-24).

[Goo17c] Google. *Certificate Transparency logs from April 25, 2017*. 2017. URL: https://www.certificate-transparency.org/ (retrieved 2019-09-24).

[Gus+17] J. Gustafsson, G. Overier, M. Arlitt, and N. Carlsson. "A First Look at the CT Landscape: Certificate Transparency Logs in Practice". In: *Proceedings of the 18th Passive and Active Measurement Conference*. Springer-Verlag, 2017, pp. 87–99.

[HFH16] M. Hastings, J. Fried, and N. Heninger. "Weak Keys Remain Widespread in Network Devices". In: *Proceedings of the 2016 ACM on Internet Measurement Conference – IMC '16*. ACM Press, 2016. DOI: 10.1145/2987443.2987486.

[Haz+12] C. Hazay, G. L. Mikkelsen, T. Rabin, and T. Toft. "Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting". In: *CT-RSA*. Springer. 2012, pp. 313–331.

[Hei+11] M. Heiderich, T. Frosch, M. Jensen, and T. Holz. "Crouching tiger - hidden payload: security risks of scalable vectors graphics". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011*. 2011, pp. 239–250.

[Hei+14] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk. "Scriptless attacks: Stealing more pie without touching the sill". In: *Journal of Computer Security* 22.4 (2014), pp. 567–599.

[Hen+12] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices". In: *Proceedings of the 21st USENIX Security Symposium (USENIX Security 12)*. USENIX, 2012, pp. 205–220. URL: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger.

[Her+10] J. Hermans, M. Schneider, J. Buchmann, F. Vercauteren, and B. Preneel. "Parallel Shortest Lattice Vector Enumeration on Graphics Cards". In: *Progress in Cryptology – AFRICACRYPT 2010*. Springer-Verlag, 2010, pp. 52–68.

[Hol+16] R. Holz, J. Amann, O. Mehani, M. Wachs, and M. A. Kaafar. "TLS in the Wild: An Internet-wide Analysis of TLS-based Protocols for Electronic Communication". In: *Proceedings 2016 Network and Distributed System Security Symposium*. Internet Society, 2016. DOI: 10.14722/ndss.2016.23055.

[Hol+11] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. "The SSL landscape: a thorough analysis of the x.509 PKI using active and passive measurements". In: *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, IMC '11*. 2011, pp. 427–444.

[How97] N. Howgrave-Graham. "Finding small roots of univariate modular equations revisited". In: *Crytography and Coding*. Springer Berlin Heidelberg, 1997, pp. 131–142. DOI: 10.1007/bfb0024458.

[ICA06] ICAO. "ICAO Doc 9303, Machine Readable Travel Documents". In: 2006. URL: https://www.icao.int/publications/pages/publication.aspx?docnum=9303 (retrieved 2019-09-24).

[ICS17] ICSI. *The ICSI Certificate Notary*. 2017. URL: https://notary.icsi.berkeley.edu/ (retrieved 2019-09-24).

[IEE00] IEEE. *Standard Specifications for Public-Key Cryptography*. IEEE Std 1363. 2000. DOI: 10.1109/IEEESTD.2000.92292. URL: https://books.google.com/books?id=KKc8nQAACAAJ (retrieved 2019-09-24).

[Inc+16] M. S. Inci, B. Gulmezoglu, T. Eisenbarth, and B. Sunar. "Co-location detection on the cloud". In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer-Verlag. 2016, pp. 19–34.

[Ind14] F. Indutny. *Extracting server private key using Heartbleed OpenSSL vulnerability*. GitHub. 2014. URL: https://github.com/indutny/heartbleed (retrieved 2019-09-24).

[Int14] Intel. "Intel Xeon Processor E5-2660 v3 CPU specification". In: Intel. 2014. URL: https://ark.intel.com/products/81706/Intel-Xeon-Processor-E5-2660-v3-25M-Cache-2%5C_60-GHz (retrieved 2019-09-24).

[Ira+15] G. Irazoqui, M. S. IncI, T. Eisenbarth, and B. Sunar. "Know thy neighbor: crypto library detection in cloud". In: *Proceedings on Privacy Enhancing Technologies* 2015.1 (2015), pp. 25–40.

[JSS15] T. Jager, J. Schwenk, and J. Somorovsky. "On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security – CCS '15*. ACM Press, 2015. DOI: 10.1145/2810103.2813657.

[JMR17] JMRTD. *Certificates for document validation*. 2017. URL: http://jmrtd.org/certificates.shtml (retrieved 2019-09-24).

[JP06] M. Joye and P. Paillier. "Fast Generation of Prime Numbers on Portable Devices: An Update". In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Springer-Verlag, 2006, pp. 160–173.

[JPV00] M. Joye, P. Paillier, and S. Vaudenay. "Efficient Generation of Prime Numbers". In: *Cryptographic Hardware and Embedded Systems — CHES 2000*. Springer-Verlag, 2000, pp. 340–354.

[Kal98] B. Kaliski. *RFC 2313: PKCS #1: RSA Encryption Version 1.5*. Internet Engineering Task Force (IETF). 1998. URL: https://tools.ietf.org/html/rfc2313 (retrieved 2019-09-24).

[Kam13] A. Kaminsky. *Parallel Java 2 Library (PJ2)*. 2013. URL: https://www.cs.rit.edu/~ark/pj2.shtml (retrieved 2019-09-24).

[Kel02] J. Kelsey. "Compression and Information Leakage of Plaintext". In: *Fast Software Encryption*. Springer Berlin Heidelberg, 2002, pp. 263–276. DOI: 10.1007/3-540-45661-9_21.

[Kha17] S. Khandelwal. *Serious Crypto-Flaw Lets Hackers Recover Private RSA Keys Used in Billions of Devices*. The Hacker News. 2017. URL: https://thehackernews.com/2017/10/rsa-encryption-keys.html (retrieved 2019-09-24).

[Kik14] M. Kikuchi. *How I discovered CCS Injection Vulnerability (CVE-2014-0224)*. 2014. URL: http://ccsinjection.lepidum.co.jp/blog/2014-06-05/CCS-Injection-en/index.html (retrieved 2019-09-24).

[KS15] R. Klafter and E. Swanson. *Evil 32*. 2015. URL: https://evil32.com/ (retrieved 2019-09-24).

[Kle+10] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann. "Factorization of a 768-Bit RSA Modulus". In: *Advances in Cryptology – CRYPTO 2010*. Springer Berlin Heidelberg, 2010, pp. 333–350. DOI: 10.1007/978-3-642-14623-7_18.

[KPR03] V. Klíma, O. Pokorný, and T. Rosa. "Attacking RSA-Based Sessions in SSL/TLS". In: *Cryptographic Hardware and Embedded Systems – CHES 2003*. Springer Berlin Heidelberg, 2003, pp. 426–440. DOI: 10.1007/978-3-540-45238-6_33.

[Kor09] J. D. Kornblum. “Implementing BitLocker Drive Encryption for forensic analysis”. In: *Digital Investigation* 5.3 (2009), pp. 75–84. ISSN: 1742-2876. DOI: http://dx.doi.org/10.1016/j.diin.2009.01.001. URL: http://jessekornblum.com/publications/di09.pdf.

[KB15] M. Kranch and J. Bonneau. “Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning”. In: *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*. 2015.

[Kro+17] K. Krombholz, W. Mayer, M. Schmiedecker, and E. R. Weippl. “"I Have No Idea What I’m Doing" - On the Usability of Deploying HTTPS”. In: *26th USENIX Security Symposium, USENIX Security 2017*. 2017.

[KK08] N. Kumar and V. Kumar. “Analysis of Window Vista Bitlocker Drive Encryption”. In: NVLabs. 2008. URL: http://www.nvlabs.in/uploads/projects/nvbit/nvbit_bitlocker_presentation.pdf (retrieved 2019-09-24).

[Lau17] A. Laurie. “e-passport Certificates”. In: 2017. URL: http://rfidiot.org/certificates.html (retrieved 2019-09-24).

[LH95] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Society for Industrial and Applied Mathematics, Jan. 1995. DOI: 10.1137/1.9781611971217.

[Laz+14] D. Lazar, H. Chen, X. Wang, and N. Zeldovich. “Why does cryptographic software fail?: a case study and open problems”. In: *Proceedings of 5th Asia-Pacific Workshop on Systems*. ACM. 2014, pp. 1–7.

[LSJ13] S. Lekies, B. Stock, and M. Johns. “25 million flows later: large-scale detection of DOM-based XSS”. In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS’13*. 2013, pp. 1193–1204.

[Len+12] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. “Public Keys”. In: *Advances in Cryptology - Crypto 2012*. Vol. 7417. Lecture Notes in Computer Science. Springer-Verlag, 2012, pp. 626–642.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. “Factoring polynomials with rational coefficients”. In: *Mathematische Annalen* 261.4 (1982), pp. 515–534.

[Lev17] O. Levillain. *A study of the TLS ecosystem*. Dissertation thesis. 2017. URL: https://tel.archives-ouvertes.fr/tel-01454976/document (retrieved 2019-09-24).

[LGD15] O. Levillain, B. Gourdin, and H. Debar. "TLS Record Protocol: Security Analysis and Defense-in-depth Countermeasures for HTTPS". In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security - ASIA CCS '15*. ACM Press, 2015. DOI: 10.1145/2714576.2714592.

[Ley17a] J. Leyden. *Confusion reigns over crypto vuln in Spanish electronic ID smartcards*. The Register. 2017. URL: https://www.theregister.co.uk/2017/11/15/spanish_id_card/ (retrieved 2019-09-24).

[Ley17b] J. Leyden. *ROCA 'round the lock: Gemalto says IDPrime .NET access cards bitten by TPM RSA key gremlin*. The Register. 2017. URL: https://www.theregister.co.uk/2017/10/23/roca_crypto_flaw_gemalto/ (retrieved 2019-09-24).

[LN11] D. Loebenberger and M. Nüsken. "Analyzing standards for RSA integers". In: *CoRR* abs/1104.4356 (2011). URL: http://arxiv.org/abs/1104.4356.

[LN14] D. Loebenberger and M. Nüsken. "Notions for RSA Integers". In: *International Journal of Applied Cryptography*. Inderscience Publishers, 2014, pp. 116–138.

[Lyo17] G. Lyon. *Nmap Remote OS Detection*. 2017. URL: https://nmap.org/book/osdetect.html (retrieved 2019-09-24).

[Mak12] S. Mak. "Verify dependencies using PGP". In: 2012. URL: http://branchandbound.net/blog/security/2012/08/verify-dependencies-using-pgp/ (retrieved 2019-09-24).

[MS02] I. Mantin and A. Shamir. "A Practical Attack on Broadcast RC4". In: *Fast Software Encryption*. Springer Berlin Heidelberg, 2002, pp. 152–164. DOI: 10.1007/3-540-45473-x_13.

[Mau95] U. M. Maurer. "Fast generation of prime numbers and secure public-key cryptographic parameters". In: *Journal of Cryptology* 8.3 (1995), pp. 123–155.

[Mav+17] V. Mavroudis, A. Cerulli, P. Svenda, D. Cvrcek, D. Klinec, and G. Danezis. "A Touch of Evil: High-Assurance Cryptographic Hardware from Untrusted Components". In: *to appear at 24th ACM Conference on Computer and Communications Security (CCS'2017)*. ACM, 2017. ISBN: 978-1-4503-4946-8.

[May09] A. May. "Using LLL-Reduction for Solving RSA and Factorization Problems". In: *The LLL Algorithm*. Springer Berlin Heidelberg, 2009, pp. 315–348. DOI: 10.1007/978-3-642-02295-1_10.

[MS14] C. Meyer and J. Schwenk. "SoK: Lessons Learned from SSL/TLS Attacks". In: *Information Security Applications*. Springer International Publishing, 2014, pp. 189–209. DOI: 10.1007/978-3-319-05149-9_12.

[Mey+14] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews. "Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks". In: *Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, 2014, pp. 733–748. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/meyer.

[Mic13] Microsoft. "Technet: BitLocker Overview". In: Microsoft. 2013. URL: https://technet.microsoft.com/en-us/library/hh831713(v=ws.11).aspx (retrieved 2019-09-24).

[Mic06] S. Microsystems. "Application Programming Interface Java Card Platform, Version 2.2.2". In: 2006.

[Mir12] I. Mironov. *Factoring RSA Moduli II.* 2012. URL: https://windowsontheory.org/2012/05/17/factoring-rsa-moduli-part-ii/ (retrieved 2019-09-24).

[ML15] B. Moeller and A. Langley. *RFC 7507: TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks*. Internet Engineering Task Force (IETF). 2015. URL: https://tools.ietf.org/html/rfc7507.

[MDK14] B. Möller, T. Duong, and K. Kotowicz. *This POODLE Bites: Exploiting The SSL 3.0 Fallback*. 2014. URL: https:

//www.openssl.org/~bodo/ssl-poodle.pdf (retrieved 2019-09-24).

[Nat13] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*. FIPS 186-4. 2013. DOI: 10.6028/NIST.FIPS.186-4. URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf (retrieved 2019-09-24).

[Nem+17a] M. Nemec, D. Klinec, P. Svenda, P. Sekan, and V. Matyas. "Measuring Popularity of Cryptographic Libraries in Internet-Wide Scans". In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACSAC 2017. ACM, 2017, pp. 162–175. ISBN: 978-1-4503-5345-8. DOI: 10.1145/3134600.3134612.

[Nem+17b] M. Nemec, M. Sys, P. Svenda, D. Klinec, and V. Matyas. "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli". In: *24th ACM Conference on Computer and Communications Security (CCS'2017)*. ACM, 2017, pp. 1631–1648. ISBN: 978-1-4503-4946-8.

[Net17a] Netcraft Ltd. *NetCraft April 2017 Web Server Survey*. 2017. URL: https://news.netcraft.com/archives/2017/04/21/april-2017-web-server-survey.html (retrieved 2019-09-24).

[Net17b] Netcraft Ltd. *NetCraft operating system detection*. 2017. URL: http://uptime.netcraft.com/accuracy.html%5C#os (retrieved 2019-09-24).

[New19] L. H. Newman. *HTTPS Isn't Always as Secure as It Seems*. WIRED. 2019. URL: https://www.wired.com/story/https-isnt-always-as-secure-as-it-seems/ (retrieved 2019-09-24).

[Nik+12] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. V. Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. "You Are What You Include: Large-scale Evaluation of Remote JavaScript Inclusions". In: *ACM Conference on Computer and Communications Security, CCS'12*. 2012, pp. 736–747.

[NIS13] NIST. "FIPS PUB 201-2: Personal Identity Verification (PIV) of Federal Employees and Contractors". In: NIST. 2013. URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.201-2.pdf (retrieved 2019-09-24).

[Par14] A. Parsovs. "Identity Card Key Generation in the Malicious Card Issuer Model". In: *MTAT Research seminar report*. 2014. URL: https://courses.cs.ut.ee/MTAT.07.022/2014%5C_spring/uploads/Main/arnis-report-s14.pdf (retrieved 2019-09-24).

[PH06] S. Pohlig and M. Hellman. "An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance". In: *IEEE Transactions on Information Theory* 24.1 (2006), pp. 106–110. ISSN: 0018-9448.

[Pol74] J. M. Pollard. "Theorems on factorization and primality testing". In: vol. 76. 3. Cambridge University Press, 1974, pp. 521–528.

[Pol75] J. M. Pollard. "A Monte Carlo method for factorization". In: *BIT Numerical Mathematics* 15.3 (1975), pp. 331–334.

[Pol93] J. M. Pollard. "Factoring with cubic integers". In: *The development of the number field sieve*. Springer-Verlag, 1993, pp. 4–10. ISBN: 978-3-540-57013-4.

[Pop15] A. Popov. *RFC 7465: Prohibiting RC4 Cipher Suites*. 2015. URL: https://tools.ietf.org/html/rfc7465 (retrieved 2019-09-24).

[PHG13] A. Prado, N. Harris, and Y. Gluck. *SSL, gone in 30 seconds: A BREACH beyond CRIME*. Black Hat USA 2013. 2013. URL: https://media.blackhat.com/us-13/US-13-Prado-SSL-Gone-in-30-seconds-A-BREACH-beyond-CRIME-Slides.pdf (retrieved 2019-09-24).

[Qua18] Qualys. *SSL Pulse; Monthly Scan: October 03, 2018*. 2018. URL: https://www.ssllabs.com/ssl-pulse/ (retrieved 2018-10-29).

[Rap15] Rapid7. *Sonar SSL full IPv4 scan*. 2015. URL: https://scans.io/study/sonar.ssl (retrieved 2019-09-24).

[RD09] M. Ray and S. Dispensa. *Renegotiating TLS*. 2009. URL: https://kryptera.se/Renegotiating%20TLS.pdf (retrieved 2019-09-24).

[Reb15] N. Rebours. *Batch-GCDing Github SSH Keys*. 2015. URL: https://cryptosense.com/batch-gcding-github-ssh-keys/ (retrieved 2019-09-24).

[Res18] E. Rescorla. *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3*. 2018. URL: https://tools.ietf.org/html/rfc8446 (retrieved 2019-09-24).

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[RD12] J. Rizzo and T. Duong. *The CRIME attack*. 2012. URL: https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/ (retrieved 2019-09-24).

[RKW12] F. Roesner, T. Kohno, and D. Wetherall. "Detecting and Defending Against Third-Party Tracking on the Web". In: *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012*. 2012, pp. 155–168.

[Ron+19] E. Ronen, R. Gillham, D. Genkin, A. Shamir, D. Wong, and Y. Yarom. *The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations*. To appear in the IEEE Symposium on Security and Privacy. Available online: Cryptology ePrint Archive, Report 2018/1173 https://eprint.iacr.org/2018/1173. 2019.

[RPS18] E. Ronen, K. G. Paterson, and A. Shamir. *Pseudo Constant Time Implementations of TLS Are Only Pseudo Secure*. Cryptology ePrint Archive, Report 2018/747. https://eprint.iacr.org/2018/747. 2018.

[RP15] J. de Ruiter and E. Poll. "Protocol State Fuzzing of TLS Implementations". In: *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015, pp. 193–206. ISBN: 978-1-931971-232. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter.

[Sal+08] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. *RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State*. Internet Engineering Task Force (IETF). 2008. URL: https://tools.ietf.org/html/rfc5077.

[Sch17] E. Schechter. *Next Steps Toward More Connection Security*. Google Security Blog. 2017. URL: https://security.googleblog.com/2017/04/next-steps-toward-more-connection.html (retrieved 2019-09-24).

[Sch18] E. Schechter. *A milestone for Chrome security: marking HTTP as "not secure"*. The Keyword. 2018. URL: https://www.blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/ (retrieved 2019-09-24).

[Sch04] B. Schneier. *Secrets and lies - digital security in a networked world: with new information about post-9/11 security*. Wiley, 2004. ISBN: 978-0-471-45380-2.

[SHS15] Y. Sheffer, R. Holz, and P. Saint-Andre. *RFC 7457: Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*. Internet Engineering Task Force (IETF). 2015. URL: https://tools.ietf.org/html/rfc7457.

[Shp03] I. Shparlinski. "The Insecurity of the Digital Signature Algorithm with Partially Known Nonces". In: *Cryptographic Applications of Analytic Number Theory*. Birkhäuser Basel, 2003, pp. 201–206. DOI: 10.1007/978-3-0348-8037-4_17. URL: https://doi.org/10.1007%2F978-3-0348-8037-4_17.

[SPK16] S. Sivakorn, I. Polakis, and A. D. Keromytis. "The Cracked Cookie Jar: HTTP Cookie Hijacking and the Exposure of Private Information". In: *IEEE Symposium on Security and Privacy, SP 2016*. 2016, pp. 724–742.

[Smi14] B. Smith. *POODLE applicability to TLS 1.0+*. IETF TLS mailing list. 2014. URL: https://www.ietf.org/mail-archive/web/tls/current/msg14058.html (retrieved 2019-09-24).

[Som16a] J. Somorovsky. *Curious Padding oracle in OpenSSL (CVE-2016-2107)*. On Web-Security and -Insecurity blog. 2016. URL: https://web-in-security.blogspot.com/2016/05/curious-padding-oracle-in-openssl-cve.html (retrieved 2019-09-24).

[Som16b] J. Somorovsky. "Systematic Fuzzing and Testing of TLS Libraries". In: *Proceedings of the 2016 ACM SIGSAC Con-*

ference on Computer and Communications Security – CCS'16. ACM Press, 2016. DOI: 10.1145/2976749.2978411.

[SS13] S. Son and V. Shmatikov. "The Postman Always Rings Twice: Attacking and Defending postMessage in HTML5 Websites". In: *20th Annual Network and Distributed System Security Symposium, NDSS 2013*. 2013.

[SDH16] D. Springall, Z. Durumeric, and J. A. Halderman. "Measuring the Security Harm of TLS Crypto Shortcuts". In: *Proceedings of the 2016 ACM on Internet Measurement Conference - IMC '16*. ACM, 2016. DOI: 10.1145/2987443.2987480.

[SS01] D. R. Stinson and R. Strobl. "Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates". In: *ACISP*. Vol. 1. Springer. 2001, pp. 417–434.

[Str03] T. Straub. "Efficient two party multi-prime RSA key generation". In: *Proceedings of IASTED International Conference on Communication, Network, and Information Security*. ACTA Press. 2003, pp. 100–105.

[Šve+16a] P. Švenda, M. Nemec, P. Sekan, R. Kvašňovský, D. Formánek, D. Komárek, and V. Matyáš. "The Million-Key Question – Investigating the Origins of RSA Public Keys". In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*. USENIX, 2016, pp. 893–910. ISBN: 978-1-931971-32-4.

[Šve+16b] P. Švenda, M. Nemec, P. Sekan, R. Kvašňovský, D. Formánek, D. Komárek, and V. Matyáš. "The Million-Key Question – Investigating the Origins of RSA Public Keys". In: *FI MU Report Series, FIMU-RS-2016-03*. Masaryk University, 2016, pp. 1–83.

[Syn14] Synopsys. *The Heartbleed Bug (CVE-2014-0160)*. 2014. URL: http://heartbleed.com/ (retrieved 2019-09-24).

[The16] The FPLLL development team. "fplll, a lattice reduction library". 2016. URL: https://github.com/fplll/fplll (retrieved 2019-09-24).

[Trü17] Trüb Baltic AS. "Estonian Electronic ID card application specification, EstEID v. 3.5". In: 2017. URL: http://www.

id.ee/public/TB-SPEC-EstEID-Chip-App-v3.5-20170314.pdf (retrieved 2019-09-24).

[Tru06] Trusted Computing Group. "TPM Main Specification Version 1.2, Revision 94". In: 2006. URL: https://trustedcomputinggroup.org/tpm-main-specification/ (retrieved 2019-09-24).

[Tru11] Trusted Computing Group. "TPM Main Specification Level 2 Version 1.2, Revision 116". In: 2011. URL: https://trustedcomputinggroup.org/tpm-main-specification/ (retrieved 2019-09-24).

[Tub17] C. Tubio. *PGP keydump from April 19, 2017*. 2017. URL: http://pgp.key-server.io/dump/ (retrieved 2017-04-19).

[Vah17] A. Vahtla. *Potential security risk could affect 750,000 Estonian ID cards*. ERR. 2017. URL: https://news.err.ee/616732/potential-security-risk-could-affect-750-000-estonian-id-cards (retrieved 2019-09-24).

[Val+17] L. Valenta, D. Adrian, A. Sanso, S. Cohney, J. Fried, M. Hastings, J. A. Halderman, and N. Heninger. "Measuring small subgroup attacks against Diffie-Hellman". In: *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017. DOI: 10.14722/ndss.2017.23171.

[Val+18] L. Valenta, N. Sullivan, A. Sanso, and N. Heninger. *In search of CurveSwap: Measuring elliptic curve implementations in the wild*. Cryptology ePrint Archive, Report 2018/298. https://eprint.iacr.org/2018/298. 2018.

[Van+16] B. VanderSloot, J. Amann, M. Bernhard, Z. Durumeric, M. Bailey, and J. A. Halderman. "Towards a Complete View of the Certificate Ecosystem". In: *Proceedings of the 2016 ACM on Internet Measurement Conference, IMC 2016*. 2016, pp. 543–549.

[VP15] M. Vanhoef and F. Piessens. "All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS". In: *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015, pp. 97–112. ISBN: 978-1-931971-232. URL: https://www.usenix.

org / conference / usenixsecurity15 / technical - sessions/presentation/vanhoef.

[Vau02] S. Vaudenay. "Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS..." In: *Advances in Cryptology – EUROCRYPT 2002*. Springer Berlin Heidelberg, 2002, pp. 534–545. DOI: 10.1007/3-540-46035-7_35.

[Veh18] J. Vehent. *Security/Server Side TLS (version 4.1)*. MozillaWiki. 2018. URL: https://wiki.mozilla.org/Security/Server_Side_TLS#Recommended_configurations (retrieved 2018-10-29).

[WMY18] L. Waked, M. Mannan, and A. Youssef. "To Intercept or Not to Intercept". In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security - ASIACCS '18*. ACM Press, 2018. DOI: 10.1145/3196494.3196528. URL: https://doi.org/10.1145%2F3196494.3196528.

[Wei+16] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc. "CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016.

[Wes16] M. West. *W3C Candidate Reccomendation: Mixed Content*. 2016. URL: https://www.w3.org/TR/mixed-content/ (retrieved 2019-09-24).

[Wes18] M. West. *W3C Working Draft: Content Security Policy Level 3*. 2018. URL: https://www.w3.org/TR/CSP3/ (retrieved 2019-09-24).

[Wie90] M. J. Wiener. "Cryptanalysis of short RSA secret exponents". In: *IEEE Transactions on Information Theory* 36 (1990), pp. 553–558.

[Won15] D. Wong. "Implementation of Coppersmith attack (RSA attack using lattice reductions)". 2015. URL: https://www.cryptologie.net/article/222/implementation-of-coppersmith-attack-rsa-attack-using-lattice-reductions/ (retrieved 2019-09-24).

[YY05] O. Yacobi and Y. Yacobi. "A New Related Message Attack on RSA". In: *Public Key Cryptography – PKC 2005*.

Vol. 3386. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 1–8.

[Yub17] Yubico. "PGP, Importing keys". In: 2017. URL: https://developers.yubico.com/PGP/Importing_keys.html (retrieved 2019-09-24).

[Zhe+15] X. Zheng, J. Jiang, J. Liang, H. Duan, S. Chen, T. Wan, and N. Weaver. "Cookies Lack Integrity: Real-World Implications". In: *24th USENIX Security Symposium, USENIX Security 15*. 2015, pp. 707–721.

A Author's publications

We list the author's publications relevant to the thesis.

1. Petr Svenda, Matus Nemec, Peter Sekan, Rudolf Kvasnovsky, David Formanek, David Komarek, and Vashek Matyas. "The Million-Key Question – Investigating the Origins of RSA Public Keys". In: *Proceeding of the 25th USENIX Security Symposium (USENIX Security 2016)*. USENIX, 2016. [Šve+16a]

    - Awarded Best Paper. Contributed before being enrolled in the PhD programme (24%): analysis of software libraries (open source code and data collection), visualization and analysis of distributions of the most significant bits of private and public keys, writing.
    - An extended technical report was published as [Šve+16b].

2. Matus Nemec, Dusan Klinec, Petr Svenda, Peter Sekan, and Vashek Matyas. "Measuring Popularity of Cryptographic Libraries in Internet-Wide Scans". In: *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC 2017)*. ACM, 2017. [Nem+17a]

    - Contributions (40%): design, implementation and accuracy experiments of the measurement, data collection (Certificate Transparency dataset), result processing and visualization, writing, conference presentation.
    - The source code and datasets were published with the paper at `https://github.com/crocs-muni/classifyRSAkey`.

3. Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli". In: *Proceedings of the 24th ACM Conference on Computer and Communications Security (ACM CCS 2017)*. ACM, 2017. [Nem+17b]

- Received Real-World Impact Award. Contributions (30%): equally contributed with Marek Sys to the discovery of the vulnerability and attack development. Implementation (proof of concept, parameter optimization), writing, conference presentation.

4. Stefano Calzavara, Riccardo Focardi, Matus Nemec, Alvise Rabitti, Marco Squarcina. "Postcards from the Post-HTTP World: Amplification of HTTPS Vulnerabilities in the Web Ecosystem". In: *Proceedings of the 40th IEEE Symposium on Security and Privacy (IEEE S&P 2019)*. IEEE, 2019. [Cal+19]

    - Contributions (20% by convention): Survey of attacks on TLS, attack trees, writing, part of the implementation.