With Great Power Come Great Side Channels: Statistical Timing Side-Channel Analyses with Bounded Type-1 Errors

Martin Dunsche[1], Marcel Maehren[1], Nurullah Erinola[1], Robert Merget[2], Nicolai Bissantz[1], Juraj Somorovsky[3], and Jörg Schwenk[1]

[1]Ruhr University Bochum
[2]Technology Innovation Institute
[3]Paderborn University

Abstract

Constant-time implementations are essential to guarantee the security of secret-key operations. According to Jancar et al. [42], most cryptographic developers do not use statistical tests to evaluate their implementations for timing side-channel vulnerabilities. One of the main reasons is their high unreliability due to potential false positives caused by noisy data. In this work, we address this issue and present an improved statistical evaluation methodology with a controlled type-1 error (α) that restricts false positives independently of the noise distribution. Simultaneously, we guarantee statistical power with increasing sample size. With the bounded type-1 error, the user can perform trade-offs between false positives and the size of the side channels they wish to detect. We achieve this by employing an empirical bootstrap that creates a decision rule based on the measured data.

We implement this approach in an open-source tool called RTLF and compare it with three different competitors: *Mona*, *dudect*, and *tlsfuzzer*. We further compare our results to the t-test, a commonly used statistical test for side-channel analysis. To show the applicability of our tool in real cryptographic network scenarios, we performed a quantitative analysis with local timing measurements for CBC Padding Oracle attacks, Bleichenbacher's attack, and the Lucky13 attack in 823 available versions of eleven TLS libraries. Additionally, we performed a qualitative analysis of the most recent version of each library. We find that most libraries were long-time vulnerable to at least one of the considered attacks, with side channels big enough likely to be exploitable in a LAN setting. Through the qualitative analysis based on the results of RTLF, we identified seven vulnerabilities in recent versions.

1 Introduction

Cryptographic algorithms that operate on secret data are prone to side-channel attacks. One primary example are timing side channels, where the algorithm may leak information about the secret data through their execution time. An attacker can exploit this by benchmarking execution times of known and unknown secrets.

Side Channels Are Hard To Prevent Prime examples of timing side channels are conditional branches based on secrets and secret-based memory access, which create timing differences through the CPU cache. To prevent timing side-channel attacks, developers devised different strategies, colloquially known as constant-time programming. These strategies include, for example, the replacement of branches by logical operations to ensure a constant control flow. However, developers often do not mitigate side channels systematically; according to Jancar et al. [42], two out of 20 cryptographic libraries claimed that side-channel attacks were outside their attacker model, and another ten libraries only used best-effort strategies to mitigate them. While there are different tools available to support developers [4, 5, 6, 7, 8, 11, 14, 20, 22, 25, 31, 38, 39, 49, 50, 59, 71, 83, 84, 94, 96, 97, 98, 105], according to Jancar et al. [42], these tools are often perceived as hard to use.

All previously mentioned tools rely on a well-defined leakage model [9, 30, 56] to abstractly reason about the absence of timing side channels. However, accurately modeling real-world leakages is challenging and requires intensive knowledge about underlying software and hardware layers [9]. An example of this is the underlying assumption that the CPU instructions are executed in constant time independently of their operands. This is not always the case as, for example, the `UMUL` instruction on an ARM Cortex-M3 is not constant time. The expectations of the cryptographic community on the underlying hardware keep getting further subverted by attacks like Spectre [47], Meltdown [53], and the recent Hertzbleed [95]. Hertzbleed is especially interesting in this context, as it showed that changes in the CPU frequency may depend on the values used in otherwise identical instructions. Since the CPU frequency directly influences the execution time, an attacker may be able to derive information about a secret parameter based on a timing side channel.

Measurements Are Necessary Given the complex hardware impact on timing behavior and the unreliability of source code tools, *measuring* the timing behavior of a deployed sys-

tem is indispensable. Analysis of the gathered timing samples is offered through tools like *Mona* [75], *dudect* [68], and *tlsfuzzer* [45]. However, timing measurements are often omitted in the development life cycle; according to Jancar et al. [42], five out of 20 libraries only did manual statistical runtime testing.

Statistical Terminology A statistical evaluation can test a hypothesis on two random distributions, for example, to test if a random number generator is good, one tests the hypothesis H_0 that the distribution of its outputs is identical to the uniform distribution. If a statistical test rejects H_0, this means that the outputs of the generator are not uniformly distributed. It is then assumed that hypothesis H_1 holds, which states that the two distributions are different.

In cryptographic timing measurements, we repeatedly measure the execution time of a secret-key operation under two different preconditions P_X and P_Y (e.g., padding is correct vs. padding is incorrect). We thus produce two different statistical samples $x := (x_1; \dots; x_n)$ and $y := (y_1; \dots; y_n)$. If there is a statistically significant difference between these two samples, it may be possible to learn, for a randomly chosen ciphertext, if P_X or P_Y holds. To determine if this is the case, we test the null hypothesis, which assumes that the distributions of x and y *are* equal. If H_0 is rejected, this is an indication that the two distributions may not be equal and that there may be a difference. However, since we are working with finite samples, a test of the H_0 hypothesis may fail for two reasons: wrongfully rejecting H_0 (type-1 error, *false positive*) or wrongfully accepting H_0 (type-2 error, *false negative*). The rate of correctly rejecting H_0 is referred to as statistical power (*true positive* rate).

Statistical Guarantees In general, for two unknown distributions D_X and D_Y, it is impossible to give a bound on the type-2 error for a given sample size. This is because these distributions may be arbitrarily close but still different; think of two otherwise identical distributions D_X and $D_{X+\varepsilon}$, where the values for $D_{X+\varepsilon}$ are sampled from D_X, but a small constant value ε is added to each sample. For each sample size n, we can then choose a value ε_n such that H_0 is falsely accepted.

However, it is possible to bound the type-1 error. For parametric distributions, i.e., distributions that can be characterized by a few parameters, such tests are available in standard tools. For example, if the tested distribution is 'close enough' to the normal distribution (which can be characterized by the two parameters *mean* and *variance*), a t-test can be used. Unfortunately, such parametric approaches are inadequate for timing measurements as real-world timing measurements are highly complex and cannot be sufficiently captured through parametric distributions.

There are two research directions for designing statistical tests with a bounded type-1 error rate for non-parametric distributions:

- **Bounded type-1 error for small samples:** Since sampling (not only for timing measurements) comes with a cost, it is desirable to bound the type-1 error already for small samples. This implies that as much information as possible from the samples should be used in the test.
- **Trade-off between type-1 error rate and detection rate:** For a fixed sample size, a lower type-1 error implies a lower detection rate. For example, for a given sample size of two distributions, the mean will nearly always be different; for a low type-1 error, we would make the tolerance interval (or *confidence intervals*) for this difference, which we consider to be attributable to random noise, larger, and would therefore not detect a difference if the mean of the two distributions lies in this interval. This implies that it is desirable to use the desired type-1 error rate as an input parameter to the statistical test and not hardcode it.

Tools like *Mona* [75] and *dudect* [68] aim for low error and good detection rates for small samples, but both error and detection rates have not been quantified. Moreover, we will empirically show in Section 4 that both tools may have low detection rates or unbounded type-1 errors. This leads to our first research question:

> **RQ1: Can statistical techniques be used in the context of the non-parametric distributions of timing measurements to design tests for arbitrary samples, where the type-1 error and detection rate are balanced by an input parameter?**

We answer this question in the affirmative. Any statistical method aims to maximize the detection rate (or minimize the type-2 error) for a prescribed type-1 error. *tlsfuzzer* [45], a recently released tool for timing analyses, already addresses this challenge better than *Mona* and *dudect* but still lacks effectiveness in terms of statistical power. In our approach, we use multiple quantiles to extract as much information as possible from the distributions. In order to guarantee a balance of type-1 error and detection rate, we employ an empirical bootstrap to obtain a reasonable decision rule (i.e., a rule to reject or accept H_0) based on the measurements. Specifically, the derived decision rule does not exhibit excessively conservative behavior, nor does it produce an inflated type-1 error rate (cf. Section 4). Our construction is described in Section 3 and implemented in our tool *R-Time-Leak-Finder (RTLF)*.

Timing Measurements for TLS To evaluate our approach, we chose especially noisy timing side channels – secret data operations in complex real-world TLS implementations. For TLS implementations, various exploitable timing side channels have been published [1, 2, 16, 17, 60, 62, 104, 108], and various fixes have been applied to the source code of the attacked libraries to make them 'more constant time'. These fixes were mostly applied after a vulnerability was discovered and published – the success of these fixes is not documented. This leads to our second research question:

RQ2: Do potential timing side channels still exist in TLS implementations? If not: when have they been fixed, and are these fixes stable?

We answer this research question by performing the first quantitative timing side-channel study on TLS libraries. We tested eleven different libraries, in 823 versions. We considered Bleichenbacher [12], Vaudenay Padding Oracle [92], and Lucky13 attacks [1] which triggered a large body of side-channel research [2, 13, 55, 61, 62, 72, 73, 77, 106] and thus had continuous impact on the tested libraries.

Results in TLS Libraries Our qualitative analysis of the most recent libraries revealed that a significant number of libraries exhibit vulnerabilities that are probably exploitable in a LAN setting. Our approach detected eight potential vulnerabilities across the libraries, seven of which we could manually attribute to specific locations in the source code (cf. Section 7).

Overall, a high number of TLS libraries contained remotely exploitable timing vulnerabilities throughout different versions that are present for very long periods and sometimes still not fixed. Our study shows that throughout its history, TLS is and was in a bad state regarding the analyzed vulnerabilities (cf. Section 6 and Section 8).

Contributions We make the following contributions:

- We propose a non-parametric test based on quantile differences to statistically evaluate timing measurements with a type-1 error bounded by an input parameter α (cf. Section 3.3), which we publish in an open-source tool called RTLF[1].
- We compare RTLF with different competitors by evaluating them based on artificial timing side channels (cf. Section 4)
- We present a large-scale and longitudinal timing evaluation of eleven TLS libraries in 823 versions across three vulnerabilities and discuss the results for OpenSSL in more detail (cf. Section 6 and Section 8).
- We verify our results in the newest version of the analyzed libraries through manual code review. Our tool could detect eight potential attacks, of which we could pinpoint seven to specific locations in the code, resulting in three Vaudenay Padding Oracle vulnerabilities, three Bleichenbacher vulnerabilities, and two Lucky13 vulnerabilities (cf. Section 7).

Responsible Disclosure We contacted the developers of the libraries for which we identified exploitable timing leaks in their most recent version. Mozilla is in the process of patching the reported issues for NSS. The responses from other libraries reflect the results from Jancar et al. [42], that timing side channels are typically not systematically mitigated and are not the highest priority for the developers.

[1] https://github.com/RUB-NDS/RTLF

2 Background

Timing attacks were first described by Paul Kocher in the context of private-key operations on smartcards [48]. He showed that if the naive square-and-multiply algorithm is used for exponentiation, the Hamming weight of the exponent can directly be deduced from the execution time; with more sophisticated measurements, the private key itself can be computed. Secret-data-based cache access (a prime example for the more general case of secret data access) also poses a common source for timing differences, as frequently used data will be available faster to a program than less frequently used data.

Attacker Models The attacker's position in relation to the target is important for the exploitability of side channels. The closer the attacker is to the victim, the better the attacker can gather the leaked data. A special attacker model is the *co-located attacker*, where the attacker shares the hardware with the victim, giving the attacker the ability to probe cache lines, allowing for more powerful side-channel attacks like trace attacks. While academically, the co-located attacker model has been intensively studied [18, 19, 23, 65, 107, 108], it is not considered by all implementations. For example, OpenSSL, one of the most prominent cryptographic libraries in the world, updated its security policy in 2019 to exclude side-channel attacks with a co-located attacker from their security model. Such security issues will be mitigated, but not awarded with a CVE anymore. In our paper, we therefore focus on side channels that likely do not require a co-located attacker.

2.1 Transport Layer Security

The Transport Layer Security protocol (TLS) [28, 69] provides confidentiality, authenticity, and integrity of the exchanged data on the underlying transport protocol. To guarantee secure encryption using the record layer, all required cryptographic parameters, such as keys and algorithms, are negotiated through the TLS handshake protocol.

CBC Padding Oracle Attacks In 2002, Vaudenay first introduced the concept of a Cipher Block Chaining (CBC) Padding Oracle attack [92]. The attack relies on a decryption oracle that responds to queries about CBC padding validity. This behavior can be used to execute an adaptive chosen-ciphertext attack. TLS 1.0, 1.1, and 1.2 employ CBC mode in a MAC-then-Pad-then-Encrypt scheme [28], making them potentially vulnerable to these attacks.

Lucky13 In 2013, AlFardan and Paterson discovered Lucky13, the most commonly known timing side-channel attack on the MAC-then-Pad-then-Encrypt scheme in TLS [1]. They showed that certain padding modifications also affect the length of data validated by the MAC algorithm, resulting in a tiny timing side channel that can be exploited to run Padding Oracle attacks.

Bleichenbacher Attacks In 1998, Daniel Bleichenbacher published an adaptive chosen-ciphertext attack on RSA encryption schemes that use PKCS #1 v1.5 padding [12]. For TLS, this attack enables the attacker to recover the *Premaster Secret* of a recorded session and thus decrypt any application data exchanged by the communicating peers. Over time, different types of oracles re-enabled Bleichenbacher's attack. For example, Klima et al. [46] found an oracle resulting from an invalid TLS version in the decrypted ciphertext.

The most recent version TLS 1.3 [69] solves many of these issues by prohibiting RSA PKCS#1 v1.5 and CBC. However, Jager et al. [41] showed that a cross-protocol Bleichenbacher attack is possible against TLS 1.3, if server implementations use the same certificate for previous versions of TLS.

2.2 Statistics

Statistical hypothesis tests are used to derive a quantitative decision based on data $x := (x_1; :::; x_n)$ and $y := (y_1; :::; y_n)$. They derive a decision to accept or reject the null hypothesis H_0 in favor of an alternative hypothesis H_1 based on a *summary statistic* $S := S(x; y)$, which derives a summarizing quantity based on the data (e.g., mean). Here, $S(\ ;\ )$ is also called the *test statistic*.

Background Usually, a two-sample hypothesis test is constructed in a way that given data x and y, they select H_1 in favor of H_0 if the summary statistic $S(x; y)$ exceeds some threshold c, above which S seems at odds with H_0. Since the data contains some unpredictable randomness, there is always a risk of a false decision *against* or *for* H_0. In statistical terms, rejecting H_0 when it holds true is usually referred to as a type-1 error (*false positive*), while not rejecting the hypothesis H_0, which does not hold true, is called a type-2 error (*false negative*). In this context, α represents the predetermined type-1 error rate that is deemed acceptable. Specifically, we have $\mathsf{P}(S > cjH_0; \text{is true}) \quad \alpha$. The statistical power of a hypothesis test is the probability of rejecting H_0 when H_1 is true (i.e., the probability of a true positive). As the sample size increases, a well-designed hypothesis test should eventually reject H_0 when H_1 is indeed true. In other words, $\lim_{n\,!\,\infty} \mathsf{P}(S > cjH_1; \text{is true}) = 1$. This is an important characterization for the type-2 error, as it is asymptotically negligible in the sample size n.

Bootstrap In a lot of statistical frameworks, the threshold c may be unknown. In such cases, the true theoretical threshold c can be estimated through a procedure known as *bootstrapping*, which is based on a simulation of the test statistic for artificially generated random data with a distribution closely similar to the true but unknown distribution of the data. In this paper, we use a specific resampling strategy that depends on the data set. By repeatedly resampling the data set and imitating the behavior of S given H_0, an empirical estimator $c \quad$ for $c := c(1 \quad \alpha)$ can be obtained. More detailed, if we consider a rejection rule of the form $S > c$ and S is resampled exactly B times, we can order the resulting values, $S_1; :::; S_B$ and output the empirical threshold $c \ := S_{(b(1 \quad \alpha)Bc)}$. In summary, bootstrapping allows one to define a *configurable* threshold. For a more theoretical introduction and statistical guarantees, we refer to [89].

Quantiles The p th quantile is a statistical measure that separates the lowest $p\%$ of the data set from the remaining $(100 \quad p)\%$ of the data. The p th quantile of a continuous real-valued random variable X can be defined by $\mathsf{P}(X \quad q_p) \quad p$ and $\mathsf{P}(X \quad q_p) \quad 1 \quad p$. In other words, the p th quantile is the value q_p 2 R such that $p\%$ of the probability mass is lower or equal to q_p and $(1 \quad p)\%$ larger.

2.3 Timing Measurements

Timing measurements, which capture the real runtime of a system, can be analyzed with statistical techniques to detect timing side channels. However, statistical techniques cannot prove the absence of certain types of side channels but can only make statistical statements. Since statistical techniques operate on actual measurements, they are limited by the ability to perform *accurate* measurements. Additionally, because statistical methods have to execute the code, only a limited number of inputs from the input space can be tested. Choosing inputs that will most likely trigger side-channel vulnerabilities is up to the user of the technique.

That said, a considerable upside of statistical techniques is that they do not require a leakage model as they interact with the system like a remote attacker would. This allows statistical techniques to potentially also find issues that are outside of the expected leakage model of the developer. Statistical methods are independent of the used programming language or hardware, which makes them generally easy to use.

Mona Timing Report[2] [75] is a statistical analysis tool that performs Crossby's box test [24] for a given set of measurements. Unless the user specifies a box, Mona Timing Report will iterate over all percentile boundaries in steps of one percent. Finally, it returns the biggest box where the measurements of two distributions did not intersect. If any box was found, the distributions are assumed to be different. If no box could be found, the distributions are assumed to be the same. Subsequently, we refer to Mona Timing Report as *Mona*.

dudect[3] [68] performs a series of Welch's t-tests [74]. It collects measurements in batches until a difference has been determined. For distributions without a timing difference, *dudect* will never terminate. Based on its first set of measurements, *dudect* determines 100 percentiles, ranging from the fastest 6.7% to the fastest 99.9%, to group future measurements into data sets of increasing size. For each data set, *dudect* conducts a separate t-test as well as a t-test based on all measurements and on the variance of all measurements,

[2] https://github.com/seecurity/mona-timing-report
[3] https://github.com/oreparaz/dudect

resulting in a total of 102 t-tests. Only if the result of one t-test with at least 10,000 data sets exceeds ten, *dudect* assumes that two distributions are different.

tlsfuzzer[4] [45] conducts a series of different statistical tests. For the final decision, four distinct tests are executed, namely the Wilcoxon signed-rank test [103], Sign test [93, Section 15.3], t-test [74], and Friedman test [34]. If any of these tests reveal a statistically significant difference among the distributions, *tlsfuzzer* asserts the presence of a difference. In addition to these tests, *tlsfuzzer* also generates confidence intervals for certain quantities, such as the mean or median, through a bootstrap procedure. However, these are not part of the decision-making process.

3 Methodology

In this work, we propose a new statistical technique based on a combination of quantile estimators with a bootstrap method for detecting subtle timing side channels. This approach allows us to obtain a configurable threshold for the type-1 error that can be used to optimize the analysis towards nuanced side channels or low false positives.

3.1 Trade-Off Between Type-1 Errors and Power

Statistical approaches in tools like *Mona* [75] or *dudect* [68] use static algorithmic methods to decide if H_0 should be accepted or rejected. There are several ways in which this can cause significant problems. For example, if the static decision rule is based on some highly conservative value, one will almost always obtain a false positive rate of 0. This might seem attractive at first glance. However, a highly conservative value also reduces the probability of detecting timing differences and thus inflicts a significant loss of statistical power. We could observe such static rules defined in *dudect*.

Another potential damage can result from statistical methods using static sizes of quantiles. For example, *Mona* formulates a decision rule based on a difference in quantiles of at least 1%. This brings several drawbacks. First, the decision rule does not account for variability in the data, i.e., the rule does not compensate for larger variances. Ignoring the compensation of larger variances inflicts inflated type-1 error for noisy data. In other words, the analyses performed by *Mona* can potentially lead to false positives if the data contains much noise. Second, *Mona* excludes any side channels that are below 1% while they may still be relevant. Specifically, for any difference below 1%, *Mona*'s type-2 error will converge to 1 for $n \to \infty$.

Given these two examples, it is clear that statistical approaches that balance the type-1 and type-2 errors can improve the outcome significantly. In particular, our approach makes a conscious (and quantifiable) decision about the amount of acceptable type-1 errors.

[4] https://github.com/tlsfuzzer/tlsfuzzer

3.2 Intuition

Our approach aims to detect systematic differences between the measurements collected under two distinct preconditions P_X and P_Y, such as two differently structured protocol messages. We base our analysis on time differences in quantiles of measurements by comparing the deciles (i.e., $10\%, 20\%, \ldots, 90\%$ quantiles). Even for identical distributions, we expect some noise and measurement imprecision, meaning the deciles will never be exactly the same. Therefore, we need a threshold to determine when the deciles of the two preconditions are different enough to conclude that the distributions are distinct. To set this threshold, we repeatedly sample two sets of measurements for each vector and compare the deciles of these sets. By repeating this step $B = 10{,}000$ times, we obtain 10,000 simulated differences for each decile and vector. For a type-1 error $\alpha = 9\%$, we discard the worst 1% of differences observed in each decile. We use the next-largest value as the tolerance of the measuring setup, as it estimates the expected noise (cf. Algorithm 1). We consider the distributions A and B to be different only if the difference in at least one decile of A and B exceeds this tolerance. The type-1 error threshold amounts to at most 9% since each of the nine deciles has an empirical type-1 error of at most 1%.

3.3 Distinguishing Distributions Dynamically

There are several statistical methods discussed in the literature that can distinguish two distributions (e.g., [74]). The primary objective of our approach is to ensure robustness against type-1 errors. Specifically, our method is designed to adhere to a pre-selected maximum type-1 error rate (α), while avoiding excessive conservatism and maintaining adequate statistical power as the sample size (n) increases. Additionally, we make no distributional assumptions on the data x and y. In particular, we consider independent and identically distributed (iid) random variables $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ with finite variances. In order to compare their distributions, we will compare their quantiles q_i^X, q_i^Y. In more detail, we consider the hypothesis

$$H_0 : |q_i^X - q_i^Y| = 0, \forall i = 1, \ldots, 9, \tag{1}$$

versus

$$H_1 : \exists i = 1, \ldots, 9 : |q_i^X - q_i^Y| \neq 0. \tag{2}$$

Note that the above construction is a natural extension of non-parametric tests, like e.g. the Wilcoxon signed-rank test [103], which essentially discriminates against distributions through their median. However, due to the complex structure of timing measurements, we explicitly allow for a more sophisticated analysis by considering more quantiles than the median. In our work, we decided to consider deciles. While there is a natural conflict between testing more quantiles and the type-1 error in this construction, our preliminary simulations suggested that this is an appropriate trade-off. In particular we point out that decreasing the number of quantiles potentially produces more false negatives, while increasing it would yield

a higher false positive rate or in our case inflict more conservative thresholds for the same α.
In the following, we use a standard quantile estimator. For data $x_1;:::;x_n$, we define the i th quantile estimator by

$$\hat{q}_i^x := \begin{cases} x_{(dnie)}; & \text{if } ni \not\in \mathbb{Z} \\ \frac{1}{2}(x_{(ni)} + x_{(ni+1)}); & \text{if } ni \in \mathbb{Z} \end{cases}; \quad (3)$$

where $x_{(i)}$ denotes the i th order statistic. To construct an appropriate test, we need to define a test statistic which can test the hypothesis in (1). For that purpose, we consider

$$\hat{Q}_i(x;y) := j\hat{q}_i^x \quad \hat{q}_i^y j : \quad (4)$$

With that in hand, H_0 will be rejected, if and only if there exists $i \in \{1;:::;9\}$ such that

$$\hat{Q}_i(x;y) > c(1 \quad \alpha=9;i) ; \quad (5)$$

where $c(1 \quad \alpha;i)$, $i \in \{1;:::;9\}$ is the critical value for the test, and α denotes the level of the test which has to be fixed in advance and is supposed to amount to the type-1 error of the test. In the following section, we will determine $c(1 \quad \alpha) := (c(1 \quad \alpha=9;i))_{i=1,:::,9}$ by a bootstrap procedure (cf. Algorithm 1). In order to avoid an inflated type-1 error, we use the Bonferroni correction [78], i.e., we set $\alpha := \alpha_1=9$, where α_1 is the desired type-1 error (e.g., 0:01;0:09).

3.4 Bootstrap

Next, we discuss the estimation of the threshold $c(1 \quad \alpha)$ by a bootstrap-based threshold $c \ (1 \quad \alpha)$. The basic idea of the bootstrap is to simulate the distributional behavior under H_0 by repeated application of the method to bootstrap datasets generated from the original data by some random procedure. Such resampling strategies have not only been shown to be effective in empirical studies (e.g., [27]) but have also proven to lead to consistent decision rules (cf. [89]). In a statistical context, Algorithm 1 is usually called *empirical bootstrap* (for further details, see [89] and Appendix A). With that in hand, we obtain a sustainable decision rule by comparing

$$\hat{Q} := (\hat{Q}_1(x;y);:::;\hat{Q}_9(x;y)) > c \left(1 \quad \frac{\alpha}{9}\right) ; \quad (6)$$

where the comparison is done decilewise. We reject H_0 if and only if at least one component is larger. The final test decision is then done by Algorithm 2.

3.5 R-Time-Leak-Finder (RTLF)

We implemented our approach in a script called RTLF[5] in R. The script takes two labeled measurements and a value α as input. While the bootstrap parameter can be set by the user, we choose $B = 10;000$ for the rest of this paper. We briefly discuss the effect of this parameter in Section 8. The script

[5] https://github.com/tls-attacker/RTLF

Algorithm 1 Threshold Bootstrap

Require: Time measurements: $x = (x_1;:::;x_n)$, $y = (y_1;:::;y_n)$, bootstrap iterations: B, type-1 error: α
Ensure: $c \ (1 \quad \alpha; j)$.

function Threshold(x, y, B, α)
  for $i = 1;:::;B$ **do**
    Sample $x_1 = sample(x;n); x_2 = sample(x;n)$, $x := (x_1;x_2)$.
    Sample $y_1 = sample(y;n); y_2 = sample(y;n)$, $y := (y_1;y_2)$.
    for $j = 1;:::;9$ **do**
      Compute $Q_{i;j}(x) = j\hat{q}_j^{x_1} \quad \hat{q}_j^{x_2} j$
      Compute $Q_{i;j}(y) = j\hat{q}_j^{y_1} \quad \hat{q}_j^{y_2} j$
    end for
  end for
  for $j = 1;:::;9$ **do**
    Sort statistics in ascending order:
    $(Q_{(1);j}(x);:::;Q_{(B);j}(x)) = sort(Q_{1;j}(x);:::;Q_{B;j}(x))$.
    $(Q_{(1);j}(y);:::;Q_{(B);j}(y)) = sort(Q_{1;j}(y);:::;Q_{B;j}(y))$.
    $c \ (1 \quad \alpha=9; j) := \max\{Q_{(b(1 \quad \alpha=9)Bc);j}(x);Q_{(b(1 \quad \alpha=9)Bc);j}(y)\}$.
  end for
  return $c \ (1 \quad \alpha) := (c \ (1 \quad \alpha=9;1);:::;c \ (1 \quad \alpha=9;9))$.
end function

Algorithm 2 Timing Test

Require: Time measurements: $x = (x_1;:::;x_n)$, $y = (y_1;:::;y_n)$, bootstrap iterations: B, type-1 error: α
Ensure: classification 0 or 1.

function Test(x, y, B, α)
  $c :=$ Threshold(x, y, B, α)
  for $j = 1;:::;9$ **do**
    Compute $\hat{Q}_j(x;y) = j\hat{q}_j^x \quad \hat{q}_j^y j$
  end for
  if there exists $j = 1;:::;9$ with $\hat{Q}_j(x;y) > c_j$ **then**
    return 1
  else
    return 0
  end if
end function

outputs $c \ (1 \quad \alpha); \hat{Q}$, and a binary vector that indicates the decision for each quantile. Here, large values of $c \ (1 \quad \alpha)$ imply large variances, which can be an indicator of noise. If one coordinate of $\hat{Q}_j$ surpasses its threshold $c \ (1 \quad \alpha; j)$, we reject H_0. Here, a larger value of the absolute difference $\hat{Q}_j \quad c \ (1 \quad \alpha; j)$ implies a larger timing difference, potentially indicating a larger side channel.

4 Ground Truth Analysis

To evaluate our statistical approach, we compared RTLF to two previous statistical tools, *dudect* and *Mona*. Additionally, we also compare RTLF to Welch's t-test [74] (t-test), which was also used in the Test Vector Leakage Assessment (TVLA) approach proposed by Goodwill et al. [37], which was later recommended for side-channel analysis by ISO/IEC 17825 [40]. We excluded *tlsfuzzer* from the comparison because it requires at least three distributions as input to perform its entire range of tests. For our comparison, we created artificial applications with well-defined timing side channels such that we could gather data with ground truth. These artifi-

Side Channel	Distribution X	Distribution Y
tail	**100%:** 20x256 N	**5%:** 19x256 N + 128 N **80%:** 20x256 N **10%:** 20x256 N + 32 N **5%:** 20x256 N + 64 N
same-mean	**100%:** 40x256 N	**50%:** 10x256 N **50%:** 70x256 N
shift-5%	**100%:** 20x256 N	**100%:** 20x256 N + 250 N
shift-1%	**100%:** 20x256 N	**100%:** 20x256 N + 50 N
shift-0.9%	**100%:** 20x256 N	**100%:** 20x256 N + 45 N
same-xy	**100%:**100x256 N + 10 R	**100%:**100x256 N + 10 R

Table 1: Description of the side channels built to model various distributions based on varying numbers of no operation instruction (N) and sampling of random numbers using C's rand() call (R).

cial timing side channels were created using ASM no operation (N) instructions and system calls to the random number generator of the OS to follow specific controllable distributions (X and Y) to show how these tools perform in different situations.

In our test setup, we let a client connect locally to a server application. The client performs measurements for two distributions, X and Y. Based on the data sent by the client, the server knows which distribution to simulate. As in previous studies [58, 68], we randomized the measurement order for X and Y to mitigate the impact of side effects and allow for the iid assumption made in Section 3.3. We collected n measurements for each distribution. This setup was chosen to simulate the same effects one might see in real applications using a mix of system calls and network operations.

Test Data Generation An overview of our created side channels is presented in Table 1. The test cases *tail* and *same-mean* simulate two different distributions, X and Y, with the same mean value. *shift-5%*, *shift-1%*, and *shift-0.9%* simulate a shift between the X and Y distributions. Test case *same-xy* is used to evaluate the theoretical type-1 error α for different sample sizes – here the distributions X and Y are identical.

Rationale Some presented distributions were specifically chosen to show shortcomings in the analyzed tools while still being plausible in real-world applications. We expect that the test case *tail* should present the weaknesses of *dudect* as it operates on the mean of the distributions. The test cases *shift-5%*, *shift-1%*, and *shift-0.9%* should reveal the weaknesses of *Mona*. We expect that *Mona* fails to detect the side channels with 1% and 0.9% shifts since its static box-test boundary ranges exclude any side channels that are below 1%.

Setup For each distribution (and experiment), we gathered up to 500,000 timing values based on CPU clock cycles. For the experiments, we used a consumer desktop PC with Intel(R) Core(TM) i7-6700 CPU 3.40GHz with 16 GB RAM running Ubuntu 20.04.2 LTS.

Side Channel	Samples	RTLF	***dudect***	***Mona***	t-test
tail $(X;Y)$	30k	59.1%	0%	68.3%	40.9%
same-mean $(X;Y)$	30k	100%	100%	100%	68.9%
shift-5% $(X;Y)$	30k	100%	98.8%	100%	14.2%
shift-1% $(X;Y)$	30k	22.6%	0.2%	25.6%	43.8%
shift-0.9% $(X;Y)$	30k	22.1%	0%	23.6%	51.8%
tail $(X;Y)$	500k	100%	98.3%	95%	55.9%
same-mean $(X;Y)$	500k	100%	100%	100%	83.4%
shift-5% $(X;Y)$	500k	100%	100%	100%	99.1%
shift-1% $(X;Y)$	500k	97.5%	50.9%	33.2%	31.6%
shift-0.9% $(X;Y)$	500k	93.5%	21.2%	12%	45.3%

Table 2: Analysis of the *true positive* rates based on the generated side channels. We evaluated each tool 1,000 times with each side channel and state. We configured RTLF and the t-test with $\alpha = 0.09$.

4.1 Evaluation

For the tool comparison, we configured RTLF with a type-1 error rate of $\alpha = 0.09$. We chose this value as it is large enough to illustrate how our type-1 error rate behaves for varying sample sizes (see Table 3), while also remaining low enough to be distinguishable from an uncontrolled type-1 error in all sample sizes. Moreover, we adapted *dudect* to operate on a given set of measurements as it was designed to collect measurements itself. We ran each tool 1,000 times with samples of the size given in Table 2. For each tool, we state how often H_0 was rejected, which corresponds to a *True Positive* for *tail*, *same-mean*, *shift-5%*, *shift-1%*, and *shift-0.9%*. For *same-xy*, in contrast, this represents a type-1 error (*False Positive*).

Result As shown in Table 2 and Table 3, our analysis confirms the expected weaknesses of *Mona* and *dudect*: For *tail*, *shift-1%*, and *shift-0.9%*, *dudect* fails to detect the existing side channels for a small sample size of 30,000. On the other hand, *dudect* shows a constant type-1 error rate of 0% for all sample sizes. Both of these effects are the result of its conservative decision rule. For *Mona*, we can see the expected drop in statistical power for a decreasingly distinct shift: for *shift-5%*, the detection rate is close to 100% for both sample sizes, while for *shift-1%* and *shift-0.9%*, we observed significantly lower detection rates, even for large sample sizes. The type-1 error, in contrast, steadily declines with increasing sample size, dropping from 40.8% (15,000 sample size) to 0% (500,000 sample size). The t-test generally performs worse than *Mona* except for *shift-1%* and *shift-0.9%*, where it is significantly better for a smaller sample size but still also does not detect the timing difference reliably. As shown in Table 3, the t-test shows a consistently high type-1 error rate of around 30% for test case *same-xy* for sample sizes up to 30,000 and only manages to remain within its α-value for large sample sizes.

In comparison, RTLF reliably detects all distinct distributions for a sample size of 500,000. For 30,000 samples,

the detection rate is on par with *Mona*. This also applies to *shift-1%* and *shift-0.9%*, where RTLF likewise failed to detect the difference reliably. Our analysis in Table 3 confirms that RTLF remains within the configured type-1 error threshold $\alpha = 0.09$, resulting in lower error rates than for *Mona* and the t-test. For completeness, we also performed a type-1 error analysis based on samples of distribution Y of the distinct side channels. The results are in line with our previous results and are shown in Table 4. We further depict the effect of varying α values on the type-1 error rate in Table 5.

Side Channel	Samples	RTLF	*dudect*	*Mona*	t-test
same-xy $(X;Y)$	15k	9.2%	0%	40.8%	27.6%
same-xy $(X;Y)$	20k	9.4%	0%	24%	30.8%
same-xy $(X;Y)$	25k	7.6%	0%	12.7%	30.6%
same-xy $(X;Y)$	30k	5.9%	0%	7.5%	30.6%
same-xy $(X;Y)$	500k	2.7%	0%	0%	8.8%

Table 3: Analysis of the type-1 error rates based on 1,000 simulations of the side channel *same-xy* with different sample sizes. We configured RTLF and t-test with $\alpha = 0.09$.

Side Channel	Samples	RTLF	*dudect*	*Mona*	t-test
tail $(Y;Y)$	30k	4.3%	0%	6.2%	8.4%
same-mean $(Y;Y)$	30k	4.3%	0%	7.1%	9.0%
shift-5% $(Y;Y)$	30k	3.5%	0%	5.1%	10.8%
shift-1% $(Y;Y)$	30k	3.4%	0%	5.9%	9.4%
shift-0.9% $(Y;Y)$	30k	3.7%	0%	5.1%	10.3%

Table 4: Analysis of type-1 error rates based on a sample size of 30,000 for measurements sampled from distribution Y of the denoted side channel. We configured RTLF and t-test with $\alpha = 0.09$.

Side Channel	Samples	$\alpha = 0.009$	$\alpha = 0.09$	$\alpha = 0.18$
tail $(Y;Y)$	30k	0.5%	4.3%	8.9%
same-mean $(Y;Y)$	30k	0.5%	4.3%	9.0%
shift-5% $(Y;Y)$	30k	0.4%	3.5%	6.7%
shift-1% $(Y;Y)$	30k	0%	3.4%	6.4%
shift-0.9% $(Y;Y)$	30k	0.5%	3.7%	7.2%

Table 5: Analysis of RTLF's type-1 error rates based on a sample size of 30,000 for measurements sampled from distribution Y of the denoted side channel.

5 Real-World Evaluation

While the previous evaluation showed that our approach performed better with artificial side channels, it is important to show that our tool also performs well in real-world applications. In this section, we discuss the real-world measurement setup and its potential noise sources. Additionally, we list all attack vectors considered in this study. We conclude the section with an interpretation guideline for the upcoming quantitative and qualitative analysis.

Collecting Measurements As our analysis required us to test a multitude of versions for eleven TLS libraries, we used the TLS-Docker-Library [55, 86], which provides Docker images for different TLS libraries. For each version, we started a docker container running the example server of the tested implementation. Where necessary and possible, we explicitly allowed RSA key exchanges to test for Bleichenbacher vulnerabilities. We collected 200,000 measurements per attack vector and measured the time until a library *reacted* to an attack vector, either by sending a TLS alert or closing the TCP connection. If a library frequently showed no reaction, we aborted the measuring process for the corresponding attack and excluded it from the study. This could, for example, happen if the library crashed or kept the TCP connection in an open but dead state. Note that this behavior is not necessarily safe.

We used TLS-Scanner [87] to automatically test if an implementation meets the requirements of the individual attacks and to adapt our measuring tool to the various legacy versions. To create and deliver the test vectors to the respective libraries, we used TLS-Attacker [77, 85], a well-established framework for a systematic analysis of TLS implementations. We spread the experiments across 21 consumer desktop PCs with Intel(R) Core(TM) i7-6700 CPUs 3.40GHz with 16 GB RAM running Ubuntu 20.04.2 LTS. In total, we evaluated 823 server versions from the TLS-Docker-Library.[6] We chose the same libraries studied by Maehren et al. [55]. However, we excluded Rustls, as it does not support RSA key exchange or CBC encryption, and s2n, as it intentionally omits alert messages and often does not close the TCP connection for error cases.

Noise The selected setup is not noise-free, with multiple potential noise sources:

1. We gather timing measurements on the same PC that generates the test vectors and that hosts the target application.
2. There are concurrent processes running on the OS.
3. The processes are not running on dedicated CPU cores.
4. We do not optimize the OS.

We deliberately chose to use a non-optimal setup to better reflect the noise in the measurements a typical user would experience when using RTLF on their own machine without special considerations.

5.1 Considered Attack Vectors

For each considered attack, we chose test vectors that account for different test cases based on previous research. Compar-

[6] https://github.com/tls-attacker/TLS-Docker-Library/releases/tag/v2.3.1

ing the timing behavior between the individual vectors of an attack indicates if a timing side channel is present.

Bleichenbacher Based on the work of Böck et al. [13], we selected the following test vectors that we send after the first server flight in a *ClientKeyExchange* message, together with a *ChangeCipherSpec* message and a *Finished* message that is encrypted under an incorrect *Master Secret*.

B1: A correctly formatted PKCS #1 v1.5 message.
B2: A PKCS #1 v1.5 message with invalid second byte (`0x17` instead of `0x02`).
B3: A correctly formatted PKCS #1 v1.5 message with an invalid TLS version set in the *Premaster Secret*.
B4: A PKCS #1 v1.5 message that contains a one-byte *Premaster Secret* instead of 48 bytes.
B5: A PKCS #1 v1.5 message without a `0x00` delimiter.

Padding Oracle Based on the work of Merget et al. [61], we selected the following test vectors that we send after completing a handshake:

P1: A TLS record with 80 bytes divided into 59 bytes of padding, MAC, and no application data, where the first padding byte has been invalidated.
P2: A TLS record with 80 bytes divided into 59 bytes of padding, MAC, and no application data, where the first byte of the MAC has been invalidated.
P3: A TLS record solely consisting of 80 `0xFF` bytes resulting in a record that is too short to parse all expected 255 padding bytes.
P4: A TLS record consisting solely of 80 padding bytes leaving no bytes for MAC and application data parsing.

Lucky13 For the Lucky13 attack, we selected test vectors as described in the distinguishing attack by AlFardan and Paterson [1] that we send after completing a handshake:

L1: A TLS record containing 287 random bytes followed by a `0x00` padding length byte, i.e., the record is perceived to contain no padding.
L2: A TLS record containing 32 random bytes followed by 256 `0xFF` bytes, i.e., the record is perceived to contain 255 bytes of valid padding.

5.2 Putting Figures Into Context

When analyzing the results of timing measurements, one has to be careful with the conclusions that are drawn from the results. Generally speaking, a statistical test identifies structural differences in the two gathered samples but can not specify the source of that difference. This means that any structural difference observed can be due to a timing side-channel vulnerability in the target or a bias in the measurement setup. Therefore, it is evident that the measurement setup should be constructed carefully to minimize technical influences that could cause a structural bias between vectors.

Color Encoding Throughout the paper, we use figures with color encoding to represent detected timing differences and their respective size. Our choice for the boundaries for the different colors is inspired by Crosby et al. [24], who mentions that side channels bigger than 100 ns (red) are likely remotely exploitable in a LAN setup and that timing differences bigger than 20;000 ns (black) are likely exploitable over the Internet. To provide further insight into our data, we introduced a third option at 500 ns (dark red) to indicate timing differences that are *very* likely exploitable. We depict timing differences below 100 ns in yellow. Green fields depict versions for which no statistically significant difference has been reported. Since the other tools considered in our comparison do not estimate the timing difference in ns, we use pink to depict their boolean findings.

Type-1 Errors One key feature of our approach is that it allows the user to define how many type-1 errors are acceptable in the results. This means that our results will also contain a certain amount of type-1 errors. However, the bigger the reported timing difference, the less likely it is indeed a false positive. Concretely, for our figures, this means that yellow squares are more likely false positives than red, dark red, or black squares. However, note that a consistent grouping of yellow squares across multiple subsequent versions is likely the result of a subtle timing difference and not solely due to type-1 errors.

Type-2 Errors While we try to provide high statistical power, we must refrain from making absolute claims regarding type-2 errors. This implies that the presence of green squares cannot guarantee a completely side-channel-free implementation. Rather, it signifies that any potential side channel might be too subtle for us to confidently assert its presence through our measurements for the given variance of a tested library and the noise introduced by the measurement setup (see Section 2.2).

6 Quantitative Analysis of OpenSSL

In this section, we examine the efficacy of RTLF. The primary emphasis is on evaluating its effectiveness for the real-world application OpenSSL and showcasing its notable enhancements when compared to the tools described in Section 2.3. To assess our results for OpenSSL, we use known protocol vulnerabilities corresponding to our considered test vectors.

Figure 1 presents the outcomes of our timing analyses conducted on OpenSSL, arguably the most important cryptographic library. To test for timing leaks, we pairwise compared the measurements of the considered vectors of an attack. Note that the project is developed in several different version branches simultaneously, meaning that patches for discovered vulnerabilities typically get backported to older versions as long as they are still maintained. Within Figure 1, we indicate simultaneous fixes to different branches using identical numbers (① to ④).

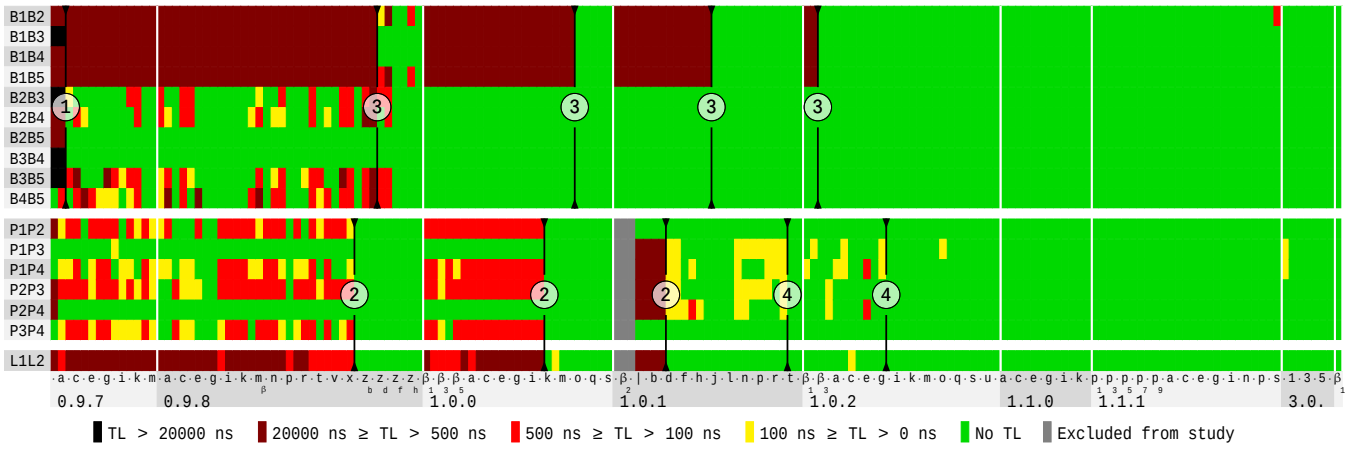
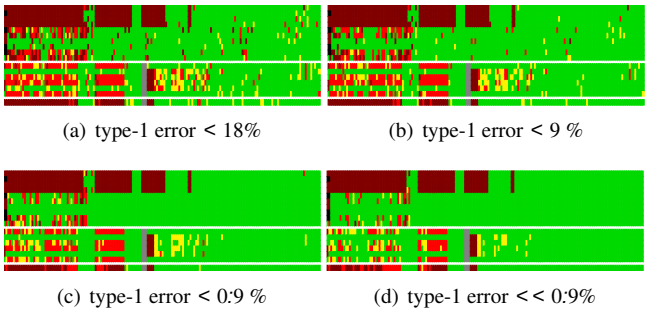


Figure 1: Result of the timing analysis for version 0.9.7 (2002) to 3.1.0-beta1 (2022) of OpenSSL. Each row shows the 17 analysis results obtained for a single version through pairwise comparison of measurements collected for Bleichenbacher (B), Padding Oracle (P), and Lucky13 (L) vectors. We categorized timing leaks (TL) based on Crosby's classification. Note that different version branches of OpenSSL are developed in parallel. The numbers indicate simultaneously deployed countermeasures against the Klima-Pokorny-Rosa (CVE-2003-0131) ①, Lucky13 (CVE-2013-0169) ②, various Bleichenbacher ③ (2014), and CVE 2016-2107 oracles ④. The labels below indicate every second version tested.

Bleichenbacher The first referenced patch (①) targeted one of the first available CVEs in OpenSSL, the Klima-Pokorny-Rosa Bleichenbacher vulnerability (CVE-2003-0131). It addressed significant timing differences in OpenSSL 0.9.7 and 0.9.7a. Accordingly, we see a transition from grave timing differences to only subtle leaks (e.g., vector pair B3B5) or no difference (B2B5 and B3B4). However, some distinct Bleichenbacher timing leaks are still present up to branches 0.9.8, 1.0.0, 1.0.1, and beta versions of 1.0.2. These have been addressed by a simultaneous patch (③) that specifically aimed to make RSA-related code constant time. As shown, for each of the affected branches, we see a transition from red to green in at least two Bleichenbacher vectors. Interestingly, we still detected more subtle timing differences throughout later versions of 0.9.8. We inspected the code after the fix and found that the developers knowingly kept some code that branches based on the PKCS padding validity, and thus OpenSSL still did not run in constant time. Interestingly, this check considers our vectors B1, B3, and B4 as compliant, while B2 and B5 are considered non-compliant. This matches our results as we only see timing differences when comparing a compliant vector with a non-compliant one (B1B2, B1B5, B2B3, B2B4, B3B5, and B4B5) but never within these two sets. We found the same code in the 1.0.0 branch after the countermeasure had been deployed at ③ but were unable to detect timing differences based on our 200,000 measurements. We tested versions from both branches with 500,000 measurements and found that we identify these timing differences more consistently in 0.9.8 and also find them sporadically in 1.0.0 versions. We assume the difference is less prevalent in branch 1.0.0 due to side effects of other parts of the code.

Padding Oracle & Lucky13 Generic Padding Oracle attacks and the Lucky13 attack are closely related as they both exploit timing leaks in the TLS record layer. As shown, deployment of the Lucky13 countermeasure (②) eliminated the timing leak in 0.9.8y and 1.0.0k. Up to this point, RTLF consistently reported timing differences for related vectors with only a few exceptions. For branch 1.0.1, the Lucky13 countermeasure eliminated the timing leak for L1L2 but only reduced the leak for P1P3, P1P4, P2P3, and P2P4. This is caused by a second-timing leak present only in early versions of 1.0.1 and 1.0.2. These versions were the first to use AES-NI. Unfortunately, there was a bug in OpenSSL that caused an observable behavior difference if a record contained overflowing padding. Since our test vectors P3 and P4 contain overflowing padding, we can observe a timing leak, but it appears to be less prevalent than the previous Lucky13 leaks. As a result, we do not observe a continuous block. Still, with the introduction of the fix (④), we do not observe further consistent timing leaks for these vectors.

Scattered Findings RTLF reported isolated unexpected timing differences. This applies to findings for Lucky13 tests in versions 1.0.0l and 1.0.2c, Padding Oracle tests in 1.0.2n and 3.0.0, and Bleichenbacher tests in 1.1.1s. Given the number of versions and vectors we evaluated, it is plausible to assume that these findings result from the configured type-1 error threshold of 0.9%, are very small side channels close to the decision rule, or bias introduced by our measurement setup. Choosing the threshold is a trade-off between low false positives and false negatives. To illustrate the effect, Figure 2 uses the same measurements as Figure 1 but presents the outcomes for varying α values. As can be seen, a higher threshold re-



(a) type-1 error < 18% (b) type-1 error < 9 %

(c) type-1 error < 0:9 % (d) type-1 error << 0:9%

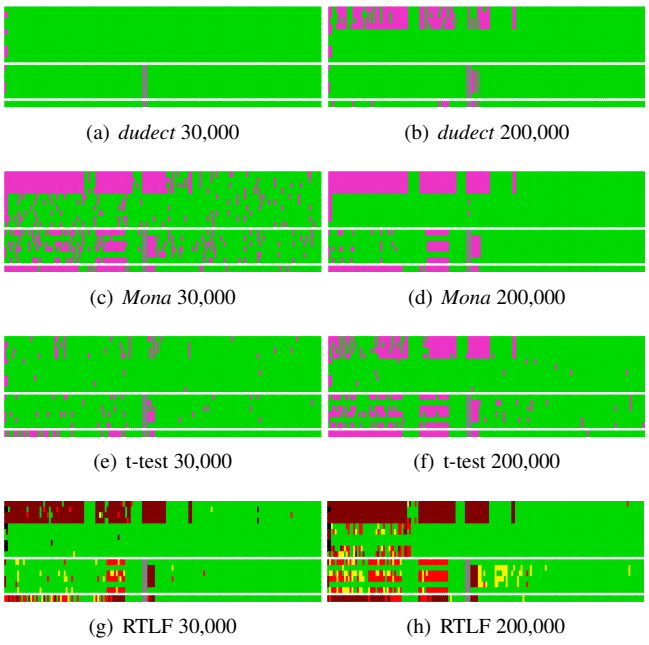
Figure 2: Overview of the effect of different type-1 error thresholds on the analysis of our OpenSSL measurements. For (d), we used the worst noise we sampled in our $B = 10{,}000$ bootstrap iterations, resulting in the lowest possible type-1 error for our collected measurements.

sults in more scattered findings. For the lowest threshold, the unexpected findings in the newer versions disappear as they are significantly less prevalent. However, some expected side channels also appear less distinct for lower thresholds.

6.1 Comparison to *Mona*, *dudect*, t-test, and *tlsfuzzer*

Mona* and *dudect In addition to our ground truth evaluation from Section 4, we also analyzed our OpenSSL measurements using *Mona* and *dudect* to assess how RTLF performs compared to these tools. We tested all three tools with 30,000 and all 200,000 measurements per vector. The results are shown in Figure 3 and demonstrate the expected shortcomings for *Mona* and *dudect* (discussed in Section 3.1) even for real-world applications like OpenSSL: *dudect* sets a conservative decision rule that causes most expected timing leaks to remain unnoticed. This is especially evident for 30,000 measurements, where only the CVE-2003-0131 timing leak (fixed at ① in Figure 1) has been detected. For 200,000 measurements, *dudect* detects most Bleichenbacher timing leaks but fails to identify Padding Oracle and Lucky13 leaks reliably. On the contrary, *Mona* detects most expected timing leaks at 30,000 measurements. However, it also produces a notably high number of scattered findings across all versions, which are often inconsistent and likely to be false positives. This issue primarily arises from the fact that *Mona*'s decision rule does not account for the variance of the quantile estimator, leading to a higher false positive rate for smaller sample sizes. For 200,000 measurements, these scattered findings disappear. Nevertheless, because the decision rule remains static, requiring a difference above the 1% threshold, smaller timing leaks, especially those related to CVE-2016-2107 (fixed at ④ in Figure 1), remain unnoticed. Compared to *Mona*, RTLF finds fewer expected timing leaks for 30,000 measurements but also produces significantly less scattered findings. This is the result of the configured type-1 error threshold of 0:9%. Setting a higher threshold α would reveal more expected timing leaks but also introduce more potential false positives as demonstrated in Figure 2. Therefore, we recommend to rather increase the sample size, since then RTLF detects even minor timing leaks, i.e., obtains asymptotically statistical power 1 (recall Section 2.2). The empirical evaluation of OpenSSL confirms that RTLF consistently yields reliable outcomes, whereas static decision rules, as employed by *Mona* and *dudect*, exhibit unreliability across all sample sizes. While *dudect* possesses a high false negative rate for both sample sizes, *Mona* produces many false positives for 30,000 measurements and some false negatives for 200,000 measurements. Even though *Mona* offers some pertinent insights, the static one-percent decision rule significantly falters when dealing with timing differences below one percent, as demonstrated in CVE-2016-2017.

t-test For the smaller sample size, the t-test fails to detect any of the expected timing differences reliably, performing only slightly better than *dudect*. While the detection rate improves significantly for the larger sample size of 200,000 measurements, the results for Bleichenbacher and Padding Oracle timing leaks are still less consistent than those of RTLF. At the same time, the t-test indicates more unexpected timing leaks than RTLF for both sample sizes.



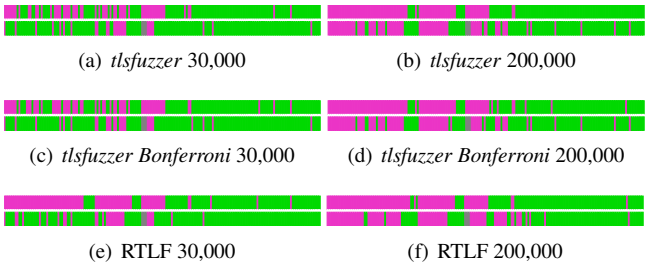
(a) *dudect* 30,000 (b) *dudect* 200,000

(c) *Mona* 30,000 (d) *Mona* 200,000

(e) t-test 30,000 (f) t-test 200,000

(g) RTLF 30,000 (h) RTLF 200,000

Figure 3: Overview of the analysis results of *dudect* (a, b), *Mona* (c, d), the t-test (e, f), and RTLF (g, h) for 30,000 measurements and 200,000 measurements each. We configured RTLF and the t-test with $\alpha = 0{:}009$. Since the t-test, *Mona* and *dudect* do not state the time difference, we use pink to depict their boolean result.



(a) *tlsfuzzer* 30,000 (b) *tlsfuzzer* 200,000

(c) *tlsfuzzer Bonferroni* 30,000 (d) *tlsfuzzer Bonferroni* 200,000

(e) RTLF 30,000 (f) RTLF 200,000

Figure 4: Overview of the analysis results of *tlsfuzzer* using the unmodified (a, b) and Bonferroni-corrected (c, d) version, in comparison to RTLF (e, f) for 30,000 measurements and 200,000 measurements each and $\alpha = 0.009$. Pink fields denote a reported timing difference. Individual attack vectors for Bleichenbacher (top) and Padding Oracle (bottom) have been merged to comply with *tlsfuzzer*'s methodology.

tlsfuzzer In contrast to *Mona*, *dudect*, and RTLF, *tlsfuzzer* comprises a combined result for all vectors of an attack. To compare RTLF to *tlsfuzzer*, we adapt our script to also determine a single result based on the measurements of all vectors of an attack. As *tlsfuzzer* requires at least three vectors to run its full range of tests, it cannot be applied to our Lucky13 tests, hence we limit our comparison to Bleichenbacher and Padding Oracle attacks. *tlsfuzzer* uses four tests and sets its type-1-error α respective for each individual test, which will not result in the same combined type-1 error, making it less conservative than RTLF with the same targeted value α. To ensure a fair comparison (i.e., test with the same type-1 error), we apply the Bonferroni correction [78] in analogy to our method. More precisely, we set the type-1 error of the four distinct tests to α=4, rather than using the correction method originally employed by the authors (in that case α=10 for 10 different Bleichenbacher vectors). Note that this adaption lowers the type-1-error α for the Friedmann test and increases α for the remaining three tests of *tlsfuzzer*. In Figure 4, we can observe that the deviations between the unmodified (a, b) and Bonferroni-corrected (c, d) versions are negligible. One can observe that while *tlsfuzzer* produces significantly better results than *dudect* and *Mona*, RTLF detects expected timing differences more consistently, especially for small sample sizes (see Figure 4(c) and 4(e)).

7 Qualitative Analysis of Newest Libraries

In this section, we provide an overview of the results of the most recent library versions of BearSSL (commit 46f7dddce75227), BoringSSL (commit d9ea5553c3c9af), Botan (2.19.3), GnuTLS (3.7.8), LibreSSL (3.7.0), MatrixSSL (4.6.0), mbedTLS (3.3.0), NSS (3.87), OpenSSL (3.0.7 & 3.1.0-beta1), tlslite-ng (0.8.0-alpha43), and wolfSSL (5.5.4). As these versions are most likely to be still in use, we increased the number of measurements to 500,000 to be able to identify smaller timing differences. Figure 5 presents the

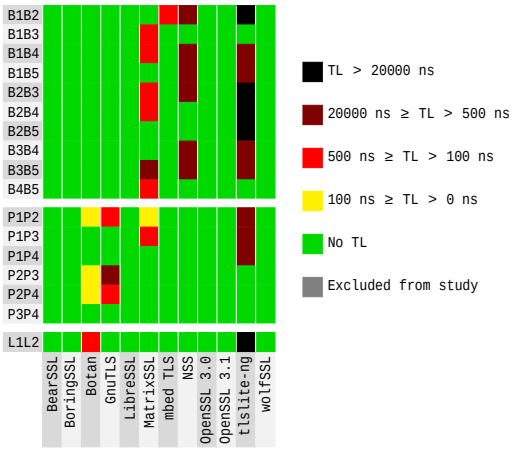


Figure 5: Results for our measurements of the most recent library versions. Each row denotes the comparison of two vectors of an attack based on the measurements collected for the library specified at the bottom of the column. We collected 500,000 measurements per attack vector and configured RTLF with $\alpha = 0.009$.

outcomes. We manually analyzed all issues for which RTLF reported a measurable side channel above 100 ns and tried to pinpoint the source of the leak.

7.1 Vulnerabilities in MatrixSSL

RTLF indicated timing differences in the measurements for Padding Oracle and Bleichenbacher vectors for MatrixSSL 4.6.0. Since the results are similar to those of other libraries, we use MatrixSSL[7] as an example for a more detailed analysis.

Bleichenbacher In Algorithm 3, we give a simplified version of MatrixSSL's PKCS #1 v1.5 parsing code used to extract the *Premaster Secret* from the padded plaintext when performing an RSA key exchange.

MatrixSSL first verifies that the initial two bytes of the padded plaintext have correct byte values. If the check fails, an error is returned (line 3). Otherwise, MatrixSSL loops over the bytes of the plaintext until it either reaches the end of the plaintext or finds the 0x00 delimiter byte (lines 6-11). It then checks if the remaining plaintext has the correct size and returns otherwise (line 14). If the size is correct, it copies the PMS to the result buffer and returns. The MatrixSSL code contains no real countermeasures against Bleichenbacher timing attacks and exits the function at multiple different positions, creating an obvious side channel. Our Bleichenbacher vectors correspond to different return calls in this function. Concretely, our vector B2 would exit at line 3, vectors B4 and B5 would exit at line 14, and B1 and B3 would exit at line 21, where the vectors B4 and B5 can additionally be separated by

[7] Note that the development of MatrixSSL has recently stopped and the repository has since been deleted.

Algorithm 3 Simplified MatrixSSL v4.6.0 PKCS#1 parsing code

Require: Decrypted PKCS byte array: *pkcs*,
Decrypt type: *decryptType* = PMS,
Expected *Premaster Secret* length: *expectedLen*

```
1:  function pkcs1UnpadExt(pkcs, expectedLen, resultBuffer)
2:      if pkcs[0] != 0x0 || pkcs[1] != 0x2 then
3:          return PS_FAILURE; //B2 will exit here
4:      end if
5:      i    1
6:      while i < len(pms) && pms[i] != 00 do
7:          if decryptType == SIGNATURE && pms[i] != 0xFF then
8:              return PS_FAILURE //this return is never taken
9:          end if
10:         i++
11:     end while
12:     i++
13:     if len(pms) - i != expectedLen then
14:         return PS_LIMIT_FAIL //B4 and B5 will exit here
15:     end if
16:     j    0
17:     while i < len(pms) do
18:         resultBuffer[j]    pms[i]
19:         i++
20:     end while
21:     return PS_SUCCESS //B1 and B3 will exit here
22: end function
```

a varying amount of loop iterations in lines 6-11.

Based on the code, we would expect to see no difference at B1B3, while for all other cases, we would expect a timing side channel, with the difference in B4B5 being the smallest.The only difference between B1, the benign message, and B3 is that B3 has an invalid TLS version set in the *Premaster Secret*. MatrixSSL, however, always overwrites this value using the protocol version at a later stage. A timing difference could thus only be caused if writing 0x0303 to the byte array is measurably faster when this value is already set. As shown in Figure 5, RTLF reported expected differences for vector comparisons B1B4, B2B3, B2B4, B3B5, and B4B5. RTLF further indicated an unexpected timing difference based on our measurements of B1 and B3. This may be a type-1 error since the executed instructions are identical, or there may be a difference in execution time caused by the concrete operands used within these instructions. Based on our measurements, RTLF did not indicate the expected differences for B1B2, B1B5, B2B5, and B3B4.

CBC Padding Oracle Regarding the Padding Oracle vectors, we found that MatrixSSL similarly has three distinct code paths for padding validation. First, MatrixSSL defines an error case when the padding bytes do not have the same value (P1). Second, MatrixSSL checks if enough bytes remain to parse a MAC, which is not possible for P3 and P4. Finally, there is a third path for when both of these checks succeed (P2). Interestingly, all three paths contain their own countermeasure against the Lucky13 attack.

Using GNU Debugger (gdb), we counted the instructions from the start of the padding and MAC validation to the return call that reports the error. For P1, we counted 68,362 instructions; for P2, we counted 68,299 instructions; and for P3 and P4, we counted 68,222 instructions. The results of RTLF align with this as we have been able to distinguish between the groups P1P2 and P1P3 but not between vectors of the same group. However, expected differences in P1P4 and P2P4 have not been detected by RTLF for a type-1 error threshold of 0.9%.

7.2 Other Libraries

Lucky13 Vulnerability in Botan RTLF indicated a timing difference in Botan for the Lucky13 vectors. In our disclosure process, the developer confirmed that the deployed countermeasure leaves a small timing difference. Again, we counted the instructions of the validation function (in a non-optimized build) using *gdb* and determined a difference of 428 instructions between the two vectors.

CBC Padding Oracle Vulnerability in GnuTLS In our analysis of GnuTLS, we discovered that the *cbc_mac_verify()* function incorporates a constant-time padding check that ensures uniformity of the padding bytes. However, the function then branches directly based on the outcome of this check. If the padding structure is correct, the function performs additional checks to ensure that there are enough bytes left after removing the padding to parse the MAC, requiring more instructions than the error case, resulting in a Padding Oracle vulnerability.

Vulnerabilities in NSS For NSS, RTLF indicated grave timing differences between Bleichenbacher vectors. Our analysis found that the code for RSA key exchanges is not written to run in constant time. NSS is mostly used in client implementations (e.g., in Mozilla's Firefox or Thunderbird), limiting the impact of this vulnerability. Mozilla has confirmed the vulnerability and is preparing a patch to address the issue.

Python Is Not Suitable for Cryptography While the code for tlslite-ng's own crypto library is written with side channels in mind, execution through Python generally does not achieve secret independent execution time. This is also pointed out by the developers in the *SECURITY.md* file on GitHub. To mitigate this, tlslite-ng can be configured to use other cryptographic libraries, such as OpenSSL. Since we measured tlslite-ng's native Python implementation, we found grave timing differences between most vectors, confirming the developers' note.

Indicated Bleichenbacher Finding in mbedTLS RTLF indicated a Bleichenbacher timing difference for mbedTLS in a single vector pair. We inspected the code and found no source for a timing difference. We conducted additional measurements, for which RTLF did not indicate a difference. We hence believe that this result was solely caused by our permitted type-1 error, which is plausible given the overall number of vectors.

Summary Based on the differences indicated by RTLF, we could identify several sources for timing differences. The only potential false positives we noticed were a single vector pair

for mbedTLS and one for MatrixSSL. These cases are indeed plausible for our prescribed α. Additionally, we identified cases where expected distinctions went unnoticed. Note that these instances can likely be attributed to the noise of the measurement setup in combination with our configured α.

7.3 Comparison to *Mona*, *dudect*, t-test, and *tlsfuzzer*

We also analyzed the measurements collected for the most recent versions using the tools from Section 2.3. To achieve comparable results, we configured the t-test and *tlsfuzzer* with $\alpha = 0.009$ as we did for RTLF. In the case of *Mona*, *dudect*, and the t-test, all identified Bleichenbacher leaks in MatrixSSL remain unnoticed. *Mona* and *dudect* additionally failed to identify any of the Padding Oracle leaks, while the t-test only missed the GnuTLS leaks. For *tlsfuzzer*, only the Bleichenbacher leaks in MatrixSSL remain unnoticed. However, *tlsfuzzer* did not produce an unexpected finding for mbedTLS, which is likely a false positive of RTLF. The t-test further indicated a Lucky13 timing leak for NSS. Based on our manual analysis of the source code, we determine this to be a false positive, which is plausible for the configured α given the number of test vectors and libraries. A graphical overview of the results is given in Figure 6 and Figure 7.

8 Discussion

Timing Differences in Older Libraries As part of our quantitative analysis of the libraries considered in Section 7, we analyzed 823 server implementations available in the TLS-Docker-Library [86]. Among the considered versions, most libraries indicate timing differences for vectors of more than one attack at some point. However, we also find that for the considered attack vectors, the most severe side channels have been mitigated in newer versions. For the timing leaks identified in Section 7, we find that these have already been present in previous versions. For Lucky13, it seems that some libraries tried to deploy countermeasures but were often times not entirely successful with their patch. We provide an extensive graphical overview of the analysis results for libraries beyond OpenSSL as part of our artifacts.[8]

Discrete Data Since we are using a non-parametric statistical test to evaluate side channels, our approach can be applied to all sorts of timing side-channels. However, challenges arise when dealing with too discrete data. This can happen, for example, if clock cycles of individual operations are measured in isolation. In this case, the estimator described in (3) may not adequately capture the differences. To address this, the estimator can be changed to one that is specifically tailored for discrete data (see Ma et al. [54] or Jentsch and Leucht [44]).

Co-Located Attackers Our results challenge the academic community's path to tackle timing side channels with static, dynamic, or symbolic analysis techniques. Without manual code review and only a limited number of vectors for each attack, we could already identify likely remotely exploitable timing differences (larger than 100 ns). Our manual investigation of the findings in the latest library versions indicates that many side channels are rather obvious, revealing a concerning gap between the academic literature and lived practice, as academia usually considers much stronger leakage models that only co-located attackers can exploit. Therefore, our results confirm the results of Jancar et al. [42] that current techniques are not widely deployed in the real world among TLS developers.

Small α Values Our methodology allows the user to control the threshold of the type-1 error. However, α can not be arbitrarily small for a fixed number of bootstrap iterations, as the threshold is tied to a percentile of the 10,000 noise simulations we perform for each decile. In principle, it is possible to choose a value for α that is greater than or equal to $1{=}B$. However, it is important to note that achieving a very small α, which is close to 0, can only be accomplished if and only if the value of $1{=}B$ is sufficiently small. Additionally, it is advisable to avoid approaching the boundary represented by $1{=}B$ as the estimation of the threshold becomes less accurate. Therefore, if a user wants smaller values for α, the bootstrap parameter B has to be adjusted as well, which increases the execution time of RTLF.

False Negatives The low type-1 error threshold comes at the cost of an increased false negative rate, which is increased by the noise in our evaluation setup. Our evaluation may also yield model-based type-2 errors when our considered attack vectors (cf. Section 5.1) are unsuitable for detecting a given vulnerability. For example, we did not detect a recent vulnerability in OpenSSL (CVE-2022-4304), which was created by a timing difference resulting from messages containing extensive leading zero bytes.[9] Since we relied on test vectors proposed by Böck et al. [13], we could not detect this vulnerability.

Fault Localization Our qualitative analysis of recent library versions required us to manually locate and verify potential findings as statistical tools are not able to point out the source of the leak. To assess a potential timing leak, a developer likewise needs to either inspect the code manually or measure isolated parts to narrow down the source. This is an inherent disadvantage, as other tools based on static, dynamic, or symbolic analysis can commonly provide fault localization [25, 98, 101, 102].

Applicability to Other Side-Channels While we demonstrated the application of RTLF to timing side channels, the approach could, in theory, be applied to other side channels like power consumption. However, RTLF is not designed for this use case, and would be computationally very inefficient.

[8] https://github.com/RUB-NDS/Artifacts-With-Great-Power-Come-Great-Side-Channels

[9] https://github.com/openssl/openssl/commit/f06ef165

Better fitting non-parametric tests like the χ^2 test have been proposed for this use case by Moradi et al. [63], which will likely perform better in this scenario.

9 Related Work

A lot of timing attacks have already been mounted on cryptographic components. Examples include timing attacks on AES [10], RSA[21], and WPA3[91]. While these attacks are traditional timing attacks that could be carried out remotely, researchers often also consider different attacker models where the attacker is co-located with the victim. This attacker model allows for powerful cache side-channel attacks [18, 23, 65, 107, 108] or even more powerful trace attacks [19].

9.1 Timing Attacks on TLS

Generic attacks on cryptographic primitives have been exploited in the context of TLS in the RSA signature generation [17] and in the ECDSA signature generation [16, 36, 76, 104]. While these attacks exploited generic flaws in the cryptographic primitives, some attacks, like the Lucky13 attack [1, 2, 73], Bleichenbacher attack [12, 13, 62, 72], Vaudenay Padding Oracle attack [61, 92], and Raccoon attack [60] are of special interest for TLS implementors; these attacks require careful considerations outside of generic cryptographic functions due to design flaws in the TLS specification itself. A technique to increase the timing signal in DTLS was proposed by Paterson and AlFardan [66]. The TIME attack [81] and HEIST attack [57] converted a compression side channel [70] into a timing side channel which reduced the required attacker model for the CRIME attack [70] to a pure Man-in-the-Browser model. While there are TLS-specific testing tools like TLS-Scanner [87], testssl.sh [82], or TLS-Anvil [55], these tools typically do not consider timing vulnerabilities.

9.2 Constant-Time Analysis

Static Techniques Static analysis techniques reason about the execution behavior of a program by statically looking at the code. This analysis can either be done for the binary itself [31], the LLVM IR [3], or the program code.[10] Static tools typically involve the annotation of secret values by the user.

Dynamic Techniques Dynamic techniques execute the code on a system and analyze the program flow. Just like static techniques, dynamic techniques typically require that secrets are annotated by the user. The dynamic analysis tool can then, for example, check that the program never branches based on a secret (or tainted) value. Dynamic analysis techniques are implemented in tools like ctgrind [50], TIMECOP [83], ct-fuzz [39], Microwalk [101], Microwalk-CI [102], and DATA [98].

[10] http://web.archive.org/web/20200810074547/http://trust-in-soft.com/tis-ct/

Symbolic Techniques A middle ground between static and dynamic analysis techniques are symbolic execution techniques, which simulate the execution of a program with symbolic inputs. This allows to reason about the reachability and potential operands of each instruction. Symbolic approaches also typically require that secrets are annotated by the user. Example tools that use the symbolic approach are CacheD [94], CaSym [15], and Binsec/Rel [25].

9.3 Statistical Techniques

Statistical analysis techniques have always been an essential point of timing evaluations, especially outside the academic community. In 2009, Crosby et al. [24] analyzed to which extent timing side-channel vulnerabilities could be measured in a remote setting. The box test proposed by Crosby et al. was implemented in *Mona* [75] and several tools presented at Blackhat [51, 58, 64]. For example, Nanown [64] uses Crosby's box test and some resampling strategies, which they call Monte Carlo bootstrapping. They support a static type-1 error rate of 5%. The method is mainly based on a heuristic approach and is less motivated by statistical theory, which makes it difficult to compare it to our proposed method from a theoretical perspective. Unfortunately, we could not empirically compare Nanown with RTLF, due to heavy IO operations caused by MySQL accesses implemented in Nanown, making evaluating large data sets time-consuming and infeasible for our comparisons.

A follow-up preprint of the *dudect* paper [68] by Fu et al. [35] also considers Welch's t-test. Fei et al. [33] implemented a theoretical framework based on likelihood methods to analyze the theoretical properties of rather general side-channel attacks in more detail. Zhang et al. [109] provided a theoretical model for cache-based side-channel attacks that also accounts for misclassifications using statistical models. As a general approach for identifying side channels with statistical techniques, Goodwill et al. proposed the Test Vector Leakage Assessment (TVLA) framework [52], which was adopted by ISO/IEC 17825 [40] for the analysis of side channels also beyond timing side channels and is commonly used to search for side-channel vulnerabilities in cryptographic primitives [43, 80, 88]. While the framework is generally flexible with the statistical test, it is most commonly used with Welch's t-test. The approach has already been tested with other statistical algorithms like the paired t-test [29] and the χ^2-test [63]. The TVLA framework is not uncontroversial and has been criticized by the community [79, 99, 100], with many of the criticisms directed at Welch's t-test and how it is applied in TVLA.

10 Conclusions & Future Work

Due to their unreliability, statistical tools have been mostly eliminated from the programming flow of many developers [42]. In this work, we showed that the perceived disadvantages of statistical techniques can be contained by providing

a configurable type-1 error α to developers, enabling them to make a trade-off between type-1 errors and how nuanced detectable side channels are. With RTLF, we give practitioners a tool specifically designed to easily detect likely remotely exploitable timing differences. This research aligns with the results from Jancar et al. [42], who states that most libraries adopt a best-effort approach to timing side channels, focusing primarily on fixing those that can be exploited remotely. Our approach is in contrast to previous mostly academic solutions, which are based on static, dynamic, and symbolic techniques that are designed to eliminate even the smallest side-channel vulnerabilities. Our results indicate a considerable gap between academic constant-time research and the lived reality of cryptographic developers who often have to prioritize other tasks. Concerningly, this results in a lot of TLS implementations containing measurable and oftentimes also likely remotely exploitable timing side-channel vulnerabilities, even though the attack vectors have been known for decades.

While our approach allowed to assert the existence of a timing difference, it is yet unclear *how* exploitable the timing difference really is as Crosby et al. [24] only give rough empirical estimates. Optimizing classification strategies under real-world conditions from an attacker's point of view may present itself as an interesting research direction in the future to more accurately model the exploitability of timing differences. Besides RTLF, there are other non-parametric two-sample tests to distinguish two distributions, such as the Kolmogorov-Smirnov test (see Darling [26]) or the Wasserstein metric (see Ramdan et al. [67]). It is of further interest, if other non-parametric testing procedures can match the performance of RTLF, or even improve it, in the context of timing side channels.

Acknowledgements

We thank the anonymous reviewers and shepherd for their valuable feedback. This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, by the Deutsche Forschungsgemeinschaft - 450197914, and by the German Federal Ministry of Education and Research (BMBF) through the project KoTeBi.

References

[1] Nadhem J. Al Fardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*.

[2] Martin R. Albrecht and Kenneth G. Paterson. Lucky microseconds: A timing attack on amazon's s2n implementation of TLS. Cryptology ePrint Archive, Paper 2015/1129.

[3] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In *Proceedings of the 25th USENIX Security Symposium*.

[4] Timos Antonopoulos, Paul Gazzillo, Michael Hicks, Eric Koskinen, Tachio Terauchi, and Shiyi Wei. Decomposition instead of self-composition for proving the absence of timing channels. *SIGPLAN Not.*, 2017.

[5] Konstantinos Athanasiou, Byron Cook, Michael Emmi, Colm MacCarthaigh, Daniel Schwartz-Narbonne, and Serdar Tasiran. Sidetrail: Verifying time-balancing of cryptosystems. In *Verified Software. Theories, Tools, and Experiments*, 2018.

[6] J. Bacelar Almeida, Manuel Barbosa, Jorge S. Pinto, and Bárbara Vieira. Formal verification of side-channel countermeasures using self-composition. *Science of Computer Programming*, Vol. 78, 2013.

[7] Gilles Barthe, Gustavo Betarte, Juan Campo, Carlos Luna, and David Pichardie. System-level non-interference for constant-time cryptography. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*.

[8] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. Computer-aided verification for mechanism design. In *Web and Internet Economics*. Springer Berlin Heidelberg, 2016.

[9] Omid Bazangani, Alexandre Iooss, Ileana Buhan, and Lejla Batina. ABBY: automating the creation of fine-grained leakage models. *IACR Cryptol. ePrint Arch.*, 2021.

[10] Daniel J. Bernstein. Cache-timing attacks on AES. 2005.

[11] Sandrine Blazy, David Pichardie, and Alix Trieu. Verifying constant-time implementations by abstract interpretation. In *Computer Security – ESORICS 2017*.

[12] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology - CRYPTO '98*.

[13] Hanno Böck, Juraj Somorovsky, and Craig Young. Return of bleichenbacher's oracle threat (ROBOT). In *27th USENIX Security Symposium*, 2018.

[14] Tegan Brennan, Seemanta Saha, Tevfik Bultan, and Corina S. Păsăreanu. Symbolic path cost analysis for side-channel detection. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018.

[15] Robert Brotzman, Shen Liu, Danfeng Zhang, Gang Tan, and Mahmut Kandemir. CaSym: Cache aware symbolic execution for side channel detection and mitigation. In *2019 IEEE Symposium on Security and Privacy (SP)*.

[16] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In Vijay Atluri and Claudia Diaz, editors, *Computer Security – ESORICS 2011*.

[17] David Brumley and Dan Boneh. Remote timing attacks are practical. In *12th USENIX Security Symposium*, 2003.

[18] Alejandro Cabrera Aldaya, Cesar Pereida García, Luis Manuel Alvarez Tapia, and Billy Bob Brumley. Cache-timing attacks on RSA key generation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019.

[19] Chen Cai-Sen, Wang Tao, Chen Xiao-Cen, and Zhou Ping. An improved trace driven instruction cache timing attack on RSA. Cryptology ePrint Archive, Paper 2011/557, 2011.

[20] Sunjay Cauligi, Gary Soeller, Fraser Brown, Brian Johannesmeyer, Yunlu Huang, Ranjit Jhala, and Deian Stefan. FaCT: A flexible, constant-time programming language. *2017 IEEE Cybersecurity Development (SecDev)*.

[21] Cai-Sen Chen, Tao Wang, and Jun-Jian Tian. An improved timing attack with error detection on RSA-CRT. Cryptology ePrint Archive, Paper 2010/054, 2010.

[22] Jia Chen, Yu Feng, and Isil Dillig. Precise detection of side-channel vulnerabilities using quantitative cartesian hoare logic. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.

[23] Shaanan Cohney, Andrew Kwong, Shahar Paz, Daniel Genkin, Nadia Heninger, Eyal Ronen, and Yuval Yarom. Pseudorandom black swans: Cache attacks on CTR_DRBG. In *2020 IEEE Symposium on Security and Privacy (SP)*.

[24] Scott A. Crosby, Dan S. Wallach, and Rudolf H. Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur.*, 2009.

[25] Lesly-Ann Daniel, Sébastien Bardin, and Tamara Rezk. Binsec/rel: Efficient relational symbolic execution for constant-time at binary-level. *CoRR*, abs/1912.08788, 2019.

[26] D. A. Darling. The kolmogorov-smirnov, cramer-von mises tests. *The Annals of Mathematical Statistics*, 28(4):823–838, 1957.

[27] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and their Application*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.

[28] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[29] A. Adam Ding, Cong Chen, and Thomas Eisenbarth. Simpler, faster, and more robust t-test based leakage detection. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 163–183, Cham, 2016. Springer International Publishing.

[30] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *J. Cryptogr. Eng.*, 1(2):123–144, 2011.

[31] Goran Doychev, Dominik Feld, Boris Köpf, Laurent Mauborgne, and Jan Reineke. CacheAudit: A tool for the static analysis of cache side channels. Cryptology ePrint Archive, Paper 2013/253, 2013.

[32] Zheng Fang and Andres Santos. Inference on directionally differentiable functions, 2016. arXiv.

[33] Yunsi Fei, A. Adam Ding, Jian Lao, and Liwei Zhang. A statistics-based fundamental model for side-channel attack analysis. Cryptology ePrint Archive, Paper 2014/152, 2014.

[34] Milton Friedman. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. *The Annals of Mathematical Statistics*, 11(1):86 – 92, 1940.

[35] Xiaohan Fu, Shuyi Ni, Siyuan Yu, Zirui Wang, and Siwei Liu. Constant-time

analysis for well-known cryptography libraries. Draft(Extended Abstract), 2021.
[36] Cesar Pereida Garcïa and Billy Bob Brumley. Constant-time callees with variable-time callers. Cryptology ePrint Archive, Paper 2016/1195, 2016.
[37] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.
[38] haybale-pitchfork. https://github.com/PLSysSec/haybale-pitchfork, Accessed: 05.05.2023.
[39] Shaobo He, Michael Emmi, and Gabriela F. Ciocarlie. ct-fuzz: Fuzzing for timing leaks. *CoRR*, abs/1904.07280, 2019.
[40] International Organization for Standardization. ISO/IEC 17825:2016, 2016. Information technology - Security techniques - Testing methods for the mitigation of non-invasive attack classes against cryptographic modules.
[41] Tibor Jager, Jörg Schwenk, and Juraj 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, page 1185–1196, New York, NY, USA, 2015. Association for Computing Machinery.
[42] Jan Jancar, Marcel Fourné, Daniel De Almeida Braga, Mohamed Sabt, Peter Schwabe, Gilles Barthe, Pierre-Alain Fouque, and Yasemin Acar. "they're not that hard to mitigate": What cryptographic library developers think about timing attacks. Cryptology ePrint Archive, Paper 2021/1650, 2021.
[43] Aruna Jayasena, Emma Andrews, and Prabhat Mishra. Tvla*: Test vector leakage assessment on hardware implementations of asymmetric cryptography algorithms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(9):1269–1279, 2023.
[44] Carsten Jentsch and Anne Leucht. Bootstrapping sample quantiles of discrete data. *Annals of the Institute of Statistical Mathematics*, 68(3):491–539, 2016.
[45] Hubert Kario. Everlasting ROBOT: the Marvin attack. Computer Security – ESORICS 2023, 2023.
[46] Vlastimil Klima, Ondrej Pokorny, and Tomas Rosa. Attacking RSA-based sessions in SSL/TLS. Cryptology ePrint Archive, Paper 2003/052, 2003.
[47] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy*, 2019.
[48] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In *CRYPTO '96, 16th Annual International Cryptology Conference*.
[49] Boris Köpf, Laurent Mauborgne, and Martín Ochoa. Automatic quantification of cache side-channels. In *Computer Aided Verification*, 2012.
[50] Adam Langley. ctgrind: Checking that functions are constant time with valgrind. https://github.com/agl/ctgrind.
[51] Nate Lawson and Taylor Nelson. Exploiting timing attacks in widespread systems, Blackhat 2010.
[52] Tal Lev-Ami and Mooly Sagiv. Tvla: A system for implementing static analyses. In Jens Palsberg, editor, *Static Analysis*, pages 280–301, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
[53] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium*, 2018.
[54] Yanyuan Ma, Marc G Genton, and Emanuel Parzen. Asymptotic properties of sample quantiles of discrete distributions. *Annals of the Institute of Statistical Mathematics*, 63:227–243, 2011.
[55] Marcel Maehren, Philipp Nieting, Sven Hebrok, Robert Merget, Juraj Somorovsky, and Jörg Schwenk. TLS-Anvil: Adapting combinatorial testing for TLS libraries. In *31st USENIX Security Symposium*, 2022.
[56] Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Inf. Secur.*, 5(2):100–110, 2011.
[57] Tom Van Goethem Mathy Vanhoef. HTTP encrypted information can be stolen through TCP-windows, August 2016.
[58] Daniel A. Mayer and Joel Sandin. Time trial, racing towards practical remote timing attacks, Blackhat 2014.
[59] MemorySanitizer. https://clang.llvm.org/docs/MemorySanitizer.html, Accessed: 05.05.2023.
[60] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. Raccoon attack: Finding and exploiting most-significant-bit-oracles in TLS-DH(E). In *Proceedings of the 30th USENIX Security Symposium*, 2021.
[61] Robert Merget, Juraj Somorovsky, Nimrod Aviram, Craig Young, Janis Fliegenschmidt, Jörg Schwenk, and Yuval Shavitt. Scalable scanning and automatic classification of TLS padding oracle vulnerabilities. In *28th USENIX Security Symposium*, 2019.
[62] Christopher Meyer, Juraj Somorovsky, Eugen Weiss, Jörg Schwenk, Sebastian Schinzel, and Erik Tews. Revisiting SSL=TLS implementations: New Bleichenbacher side channels and attacks. In *23rd USENIX Security Symposium*, 2014.
[63] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the x2-test. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 209–237, 02 2018.
[64] Timothy D. Morgan and Jason W. Morgan. Web timing attacks made practical, Blackhat 2015.
[65] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, pages 1–20, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
[66] Kenneth G. Paterson and Nadhem J. AlFardan. Plaintext-recovery attacks against datagram TLS. In *Network and Distributed System Security Symposium*, 2012.
[67] Aaditya Ramdas, Nicolás García Trillos, and Marco Cuturi. On wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, 19(2):47, 2017.
[68] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. Dude, is my code constant time? Cryptology ePrint Archive, Paper 2016/1123, 2016.
[69] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018.
[70] Juliano Rizzo and Thai Duong. The CRIME attack. In *Ekoparty Security Conference 2012*, volume 2012, 2012.
[71] Bruno Rodrigues, Fernando Magno Quintão Pereira, and Diego F. Aranha. Sparse representation of implicit flows with applications to side-channel detection. In *Proceedings of the 25th International Conference on Compiler Construction*, 2016.
[72] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. The 9 lives of Bleichenbacher's CAT: New cache attacks on TLS implementations. In *2019 IEEE Symposium on Security and Privacy*.
[73] Eyal Ronen, Kenneth G. Paterson, and Adi Shamir. Pseudo constant time implementations of TLS are only pseudo secure. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*.
[74] Lothar. Sachs. *Applied Statistics: A Handbook of Techniques*. 2 edition, 1984.
[75] Sebastian Schinzel. *Unintentional and Hidden Information Leaks in Networked Software Applications*. doctoralthesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2012.
[76] Vikram Singh. Improved timing attacks on ECDSA. Cryptology ePrint Archive, Paper 2014/636, 2014.
[77] Juraj Somorovsky. Systematic fuzzing and testing of TLS libraries. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*.
[78] SpringerLink. Bonferroni correction, 27.04.2023.
[79] François-Xavier Standaert. How (not) to use welch's t-test in side-channel security evaluations. In Begül Bilgin and Jean-Bernard Fischer, editors, *Smart Card Research and Advanced Applications*, pages 65–79, Cham, 2019. Springer International Publishing.
[80] Thomas Steinbauer, Rishub Nagpal, Robert Primas, and Stefan Mangard. TVLA on selected NIST LWC finalists. *Cryptology ePrint Archive*, 2022(887), August 2022. Cryptology ePrint Archive, Report 2022/887.
[81] Amichai Shulman Tal Be'ery. A perfect Crime? only time will tell. Blackhat Europe, March 2013.
[82] testssl.sh. https://testssl.sh/, Accessed: 05.05.2023.
[83] TIMECOP. https://www.post-apocalyptic-crypto.org/timecop/, Accessed: 05.05.2023.
[84] tis-ct. http://trust-in-soft.com/tis-ct/, Accessed: 05.05.2023.
[85] TLS-Attacker. https://github.com/tls-attacker/TLS-Attacker, Accessed: 05.05.2023.
[86] TLS-Docker-Library. https://github.com/tls-attacker/TLS-Docker-Library, Accessed: 05.05.2023.
[87] TLS-Scanner. https://github.com/tls-attacker/TLS-Scanner, Accessed: 05.05.2023.
[88] Michael Tunstall and Gilbert Goodwill. Applying tvla to public key cryptographic algorithms. *IACR Cryptol. ePrint Arch.*, 2016:513, 2016.
[89] A. W. van der Vaart. *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.
[90] Aad W. van der Vaart and Jon A. Wellner. *Weak Convergence and Empirical Processes: With Applications to Statistics*. Springer Series in Statistics. 1996.
[91] Mathy Vanhoef and Eyal Ronen. Dragonblood: Analyzing the dragonfly handshake of WPA3 and EAP-pwd. In *2020 IEEE Symposium on Security and Privacy*.
[92] Serge Vaudenay. Security flaws induced by cbc padding - applications to SSL, IPSEC, WTLS ... In *EUROCRYPT*, 2002.
[93] Dennis D. Wackerly, William Mendenhall, and Richard L. Scheaffer. *Mathematical statistics with applications*. Duxbury, 7th edition, 2007.
[94] Shuai Wang, Pei Wang, Xiao Liu, Danfeng Zhang, and Dinghao Wu. CacheD: Identifying cache-based timing channels in production software. In *Proceedings of the 26th USENIX Security Symposium*, 2017.
[95] Yingchen Wang, Riccardo Paccagnella, Elizabeth He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86. In *Proceedings of the USENIX Security Symposium*, 2022.
[96] Conrad Watt, John Renner, Natalie Popescu, Sunjay Cauligi, and Deian Stefan. Ct-wasm: Type-driven secure cryptography for the web ecosystem. *Proc. ACM Program. Lang.*, 3, 2019.
[97] Samuel Weiser, David Schrammel, Lukas Bodner, and Raphael Spreitzer. Big numbers - big troubles: Systematically analyzing nonce leakage in (EC)DSA implementations. In *Proceedings of the 29th USENIX Security Symposium*,

2020.

[98] Samuel Weiser, Andreas Zankl, Raphael Spreitzer, Katja Miller, Stefan Mangard, and Georg Sigl. DATA–differential address trace analysis: Finding address-based side-channels in binaries. In *Proceedings of the 27th USENIX Security Symposium*, 2018.

[99] Carolyn Whitnall and Elisabeth Oswald. A cautionary note regarding the usage of leakage detection tests in security evaluation. Cryptology ePrint Archive, Paper 2019/703, 2019. https://eprint.iacr.org/2019/703.

[100] Carolyn Whitnall and Elisabeth Oswald. A critical analysis of iso 17825 (‘testing methods for the mitigation of non-invasive attack classes against cryptographic modules’). 11923:256–284, 2019.

[101] Jan Wichelmann, Ahmad Moghimi, Thomas Eisenbarth, and Berk Sunar. MicroWalk: A framework for finding side channels in binaries. *CoRR*, abs/1808.05575, 2018.

[102] Jan Wichelmann, Florian Sieck, Anna Pätschke, and Thomas Eisenbarth. Microwalk-CI. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*.

[103] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[104] David Wong. Timing and lattice attacks on a remote ECDSA OpenSSL server: How practical are they really? Cryptology ePrint Archive, Paper 2015/839, 2015.

[105] Meng Wu, Shengjian Guo, Patrick Schaumont, and Chao Wang. Eliminating timing side-channel leaks using program repair. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018.

[106] Yuan Xiao, Mengyuan Li, Sanchuan Chen, and Yinqian Zhang. Stacco: Differentially analyzing side-channel traces for detecting ssl/tls vulnerabilities in secure enclaves. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.

[107] Zhao Xin-jie, Wang Tao, and Zheng Yuan-yuan. Cache timing attacks on camellia block cipher. Cryptology ePrint Archive, Paper 2009/354, 2009.

[108] Yuval Yarom, Daniel Genkin, and Nadia Heninger. CacheBleed: a timing attack on OpenSSL constant-time RSA. *Journal of Cryptographic Engineering*, 7, 06 2017.

[109] Liwei Zhang, A Adam Ding, Yunsi Fei, and Zhen Hang Jiang. Statistical analysis for access-driven cache attacks against AES. *Cryptology ePrint Archive*, 2016.

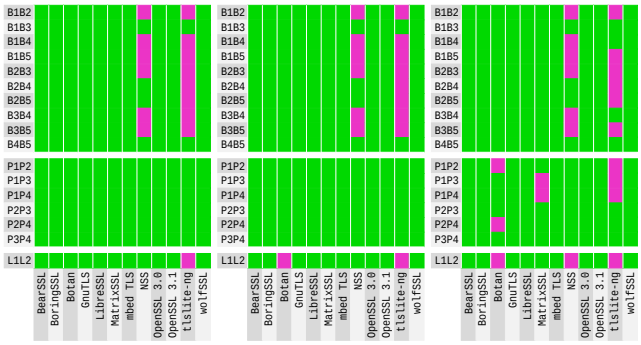
Appendix

A Theoretical Foundations

We now outline a theoretical justification for the bootstrap procedure proposed in this paper. When dealing with continuous data, we can proceed along the lines of Example 3.9.21 in van der Vaart and Wellner [90] to yield asymptotic normality using an appropriate scaling of the statistics. In addition, the Delta method for the empirical bootstrap (cf. van der Vaart [89]) can be applied to obtain the limit for the difference $\hat{q}_i^X \quad \hat{q}_i^Y$ of the quantiles estimator. However, since the Delta method for the empirical bootstrap requires Hadamard differentiable functions and the absolute value $g(x) = jxj$ is only directionally Hadamard differentiable in 0, we employ a different Delta method for $g(x)$. In particular, we refer to Theorem 2.1 and Theorem 3.3 in Fang and Santos [32], which provide an analogous asymptotic result for functions which are directionally Hadamard differentiable. However, this only guarantees α as an upper bound for the empirical bootstrap quantiles and not equality (see Theorem 3.3).

Finally, we discuss the case of discrete data, such as clock cycles. Here, the theory is more complicated since the quantile function is not Hadamard differentiable, which makes the lines of a proof sketched for continuous data in the previous paragraph invalid. However, it is worth noting that in our case (where we measure clock cycles), the empirical results did not indicate any major drawbacks in practice (compare Table 2). For a more comprehensive analysis of the asymptotic behavior for discrete data, we refer to Jentsch and Leucht [44], who present a methodology to overcome this technical difficulty through mid-distribution quantiles.

B Result Comparison for Qualitative Analysis



(a) *dudect* (b) *Mona* (c) t-test

Figure 6: Overview of the analysis results of *dudect* (a), *Mona* (b), and t-test (c) for 500,000 measurements per vector obtained using the most recent library versions. For the t-test, we set $\alpha = 0.009$, consistent with the RTLF analysis in Figure 5. Since the t-test, *Mona*, and *dudect* do not state the time difference, we use pink to depict their boolean result. With the t-test, *Mona*, and *dudect*, all Bleichenbacher leaks in MatrixSSL remained unnoticed. *Mona* and *dudect* also failed to identify any Padding Oracle leaks in the newest versions, while the t-test only missed the finding for GnuTLS. Only the t-test indicated a Lucky13 timing leak for NSS. We manually analyzed the code and could not identify any disparity in the instructions executed for the two test vectors. We thus consider it a false positive, which is also plausible for our choice of α.

(a) *tlsfuzzer* (b) RTLF

Figure 7: Overview of the analysis results of *tlsfuzzer* (a) and RTLF (b) for 500,000 measurements per vector obtained using the most recent library versions. We configured both tools with $\alpha = 0.009$. To comply with *tlsfuzzer*'s methodology, individual attack vectors have been merged. For *tlsfuzzer*, only the Bleichenbacher leaks in MatrixSSL remain unnoticed. Both versions of *tlsfuzzer* discussed in Section 6.1 produced the same results.