

Reverse-Bleichenbacher: Bleichenbacher Attacks are Practical Again

Youheng Lü¹ Lorenz Hetterich² Riccardo Zanotto^{2,3}
Cas Cremers² Michael Schwarz² Lukas Gerlach²

¹Zellic

²CISPA Helmholtz Center for Information Security

³Saarland University

V1.0, October 8, 2026.

Abstract

Bleichenbacher’s 1998 padding oracle attack against RSA PKCS#1 v1.5 has resurfaced repeatedly over the past decades, especially in TLS. Although modern browsers prefer TLS 1.3, many servers and clients still permit RSA key exchange for backward compatibility, enabling active network attackers to decrypt recorded sessions. Still, practical man-in-the-middle attacks remain challenging. The classical attack narrows the plaintext interval from the top down, spending many queries on secret-independent padding bytes before reaching the PreMasterSecret in the low-order 46 bytes of the RSA plaintext. As a result, prior attacks required multiple vulnerable servers in parallel to avoid connection timeouts.

In this paper, we present Reverse-Bleichenbacher, a novel variant of the Bleichenbacher algorithm. In contrast to all previous Bleichenbacher-style algorithms, our algorithm recovers the least significant bytes first and terminates the search as soon as the target secret is recovered. When the secret occupies only a fraction of the plaintext, as with the TLS PreMasterSecret, this significantly reduces the number of required queries. Together with improved trimmers and a coverage-optimized search, this reduces the median number of oracle queries from 11 428 to 1801 against a standards-conforming PKCS#1 v1.5 oracle with 2048-bit keys, with an overall success rate of 84.8 %. Crucially, under realistic single-server conditions (a 2.6 ms median round-trip time we observe to vulnerable hosts and 2 round trips per query) 68.3 % of all runs finish within a 30 s timeout, compared with none of the prior algorithms. To remain feasible at higher latency, we parallelize independent queries: at 20 ms round-trip time, batching reduces the median end-to-end runtime from 59.3 s to 8.3 s, at the cost of additional queries.

We identify 640 vulnerable domains in the Tranco Top 1 Million, many behind Alibaba Cloud load balancers. We demonstrate end-to-end impact on real software distribution channels: a man-in-the-middle attacker can inject malicious updates and achieve remote code execution on a victim device.

1 Introduction

Bleichenbacher’s adaptive chosen-ciphertext attack against RSA PKCS#1 v1.5 [1] is one of the most persistent cryptographic attacks in the TLS ecosystem. Since its introduction in 1998, it has repeatedly resurfaced in real-world systems, despite decades of countermeasures and protocol evolution [2, 3, 4]. A long line of follow-up work has demonstrated that padding oracle vulnerabilities continue to appear across implementations and settings [5, 6]. Notable examples include the DROWN attack [2], which exploits legacy SSLv2 support to break modern TLS connections, and ROBOT [3], which showed that Bleichenbacher-style vulnerabilities still affect widely-deployed servers. Large-scale measurement studies further confirm that padding oracles remain prevalent in the wild and are difficult to eliminate reliably [7].

TLS provides an especially important setting for these attacks. In TLS 1.0–1.2, RSA key exchange encrypts the premaster secret under the server’s RSA public key [8]. Once an attacker recovers this value, they can derive the session keys and break the confidentiality and integrity of the connection. Although TLS 1.3 removed RSA key exchange entirely [9], backward compatibility keeps it alive in deployed systems. Modern browsers still negotiate older protocol versions when necessary, and many servers continue to offer RSA-based ciphersuites [10, 11]. Consequently, a network attacker who can induce a downgrade to RSA key exchange can still turn a padding oracle into a man-in-the-middle attack [9]. Such downgrade scenarios can re-enable deprecated attack surfaces [2].

The main practical limitation is time. Classical Bleichenbacher-style attacks require many sequential oracle queries, and online attacks against browsers must finish before the handshake times out. Prior work has shown that this requirement is difficult to satisfy against a single vulnerable server, especially for TLS-relevant 2048-bit RSA keys and standards-conforming padding oracles [12, 13]. In practice, this has limited the attack to unusually favorable settings or to scenarios where multiple oracle servers can be queried in parallel [12]. This gap between theoretical vulnerability and practical exploitability has contributed to the perception that RSA downgrade attacks are largely of historical interest, even though the underlying weaknesses remain.

The main reason existing attacks are inefficient is that classical attacks learn the plaintext from the most significant bits downward [1]. In several scenarios, including TLS, this is a poor fit, as an attacker does not care about most of the PKCS#1 v1.5 structure and padding. The only value needed is the (premaster) secret at the least significant end of the plaintext. Nevertheless, the standard attack spends most of its effort narrowing the interval until it eventually reaches these final bytes. Thus, the attack wastes many queries on information the attacker does not need.

In this paper, we revisit Bleichenbacher’s attack from the opposite direction. We present Reverse-Bleichenbacher, a novel algorithm that recovers the least significant bits of the plaintext first and stops as soon as the target secret has been obtained, rather than spending queries on recovering the padding. Our approach consists of three parts. The first two closely mirror steps in the original attack and prepare the ground for our new contribution. First, we use *Trimmers* [6] to divide out small factors and prepare the message for bottom-up recovery. Second, we refine the *Searching* phase with optimizations that reduce stretching, and we incorporate information from the trimming phase to perform a finer-grained search, requiring fewer queries to obtain a new, narrower interval. Third, we introduce our novel *Bottom-Up* algorithm. The main idea is that modular multiplication by 2^{-i} acts like a “bit rotate left”: it moves the least significant bits into the most significant positions, where they can be tested with a classical oracle query. An important feature of our algorithm is that bottom-up queries can be prepared independently and processed in batches. This lets us optimize not only the total number of queries but also the number of sequential rounds, making MitM attacks feasible in realistic network settings where latency is the bottleneck. The attack requires finding a “useful” trimmer τ , which succeeds in 89% of runs. In Section 6.3 we discuss in detail the failure reasons for our algorithm, and quantify the effect of trimmers and oracle false negative rates on success probability.

Evaluation scope. Following the standard taxonomy, we model oracle types according to the specific padding checks they perform. We mainly focus on two extremes, namely *TTT* and *FFT* oracles. We give the full definitions, introduce intermediate types, and discuss oracle prevalence in Section 2.3.

In Table 1 we give a historical comparison of our results against prior work for 1024-bit keys, the only key size Bardou et al. [6] report. We then follow Capol [13] in considering also 2048-bit keys (Table 2), as this is now the dominant RSA key size in deployed TLS: a recent survey of 11 500 000 public keys found only 1% still using 1024-bit moduli [14].

Table 1: Historical comparison of median queries for 1024-bit keys. Prior-work values are from Bardou et al. [6] and Capol [13]; Table 2 extends the Capol comparison to 2048 bits.

Oracle	Original	Bardou	Capol	Ours
TTT	22 641	3768	3192	760
TFT	24 926	4014	3590	813
FFT	163 183	14 501	15 933	4152

Our evaluation shows that the resulting improvement is large enough to change the practical picture. In the 2048-bit FFT setting, we reduce the median number of oracle queries from 11 428 to 1801, corresponding to a $6.35\times$ improvement over the previous state of the art [13]. Across 100 000 simulations, the attack succeeds in 84.8 % of runs in this setting, and in 89.0 % for the permissive TTT oracle. At a measured 2.6 ms median round-trip time to vulnerable hosts and 2 round-trips per oracle query, 68.3 % of all FFT runs finish within a 30 s timeout on one server, compared with 0 % for prior work (Figure 4). At 20 ms round-trip time, batching reduces the median end-to-end runtime from 59.3 s to 8.3 s.

To assess the real-world attack surface, we scan the Tranco Top 1 Million domains [15]. We find 640 vulnerable domains, consistent with prior large-scale studies that show padding-oracle vulnerabilities persist in modern deployments [7]. We further identify a particularly concerning deployment pattern on Alibaba Cloud, where vulnerable load balancers expose padding oracles to hosted applications. Among the affected services are BigNox and LDPlayer, two widely used Android emulators. We show that their update channels can be targeted via this infrastructure. For BigNox we demonstrate an end-to-end attack in which a network adversary injects a malicious update and obtains remote code execution on the victim device.

These results show that Bleichenbacher’s attack is not a legacy protocol artifact. As long as RSA key exchange remains supported and padding oracles exist in production systems, improved cryptanalysis can turn a seemingly marginal residual attack surface into a practical threat. Our findings, therefore, argue for finally removing RSA key exchange from reachable deployments rather than relying on the assumption that the attack is too slow to matter.

Contributions. We summarize our contributions as follows.

- We present Reverse-Bleichenbacher, a novel bottom-up variant of the Bleichenbacher attack that recovers the least significant bits of the plaintext first and terminates once the target secret is found.
- We reduce the median number of oracle queries across all evaluated oracle types and key sizes (Table 2).
- We parallelize independent queries and evaluate end-to-end feasibility under different network latency conditions.
- We assess practical feasibility by scanning the Tranco Top 1M for servers vulnerable to the Bleichenbacher attack, measuring RSA cipher support and TLS handshake timeouts in popular clients, and measuring latency from nearby cloud regions to affected hosts.
- We present case studies on Alibaba Cloud load balancers and the update channels of BigNox and LDPlayer. We show that a freshly provisioned Alibaba Cloud load balancer exposes the vulnerability on default configurations and, for BigNox, demonstrate malicious update injection leading to remote code execution on a victim device.

Artifact. The artifact can be accessed through <https://github.com/cispa/ReverseBleichenbacher>

Responsible Disclosure. We responsibly disclosed our findings to affected parties. We provide more details and a disclosure timeline in Section 11.

Outline. We introduce the background in Section 2 and define our threat model in Section 3. We present our algorithmic improvements in Section 4 and describe batching and latency-aware

parallelization in Section 5. We evaluate the attack in simulation and controlled real-world settings in Section 6, present end-to-end case studies in Section 7, and measure the current Internet-wide attack surface in Section 8. We discuss broader implications in Section 9 and conclude in Section 10.

In the appendices, we describe our use of generative AI in Appendix A and describe our reproducibility artifacts in Appendix B. We revisit mitigations in Appendix C, document the Alibaba Cloud load balancer’s oracle type in Appendix D, and report additional timeout feasibility results in Appendix E. We analyze trimmer behavior in Appendix F, detail the Searching optimizations in Appendix G, and provide proofs in Appendix H. We provide a notational reference in Appendix I.

2 Background

In this section, we introduce the required background and establish notation for the rest of the paper. In particular, we fix the notation for RSA and PKCS#1 v1.5, describe the original Bleichenbacher attack, and report state-of-the-art optimizations.

2.1 RSA and PKCS#1 v1.5 Padding

RSA encryption [16] operates on integers modulo $N = pq$, where p and q are large primes; we denote the byte length of the modulus N by κ . The public key is (N, e) , where e is an integer, typically $e = 65537$; the private key is d , satisfying $ed \equiv 1 \pmod{\phi(N)}$. Given a message m represented as an integer in $[0, N)$, encryption computes $c = m^e \bmod N$ and decryption recovers $m = c^d \bmod N$. Crucially, RSA is *malleable*. Given a ciphertext c encrypting m , an attacker can choose any s and compute $\tilde{c} = c \cdot s^e \bmod N$, which decrypts to $\tilde{m} = m \cdot s \bmod N$.

To prevent this malleability from becoming a security issue, RSA encryption must be *padded*. The main RSA encryption padding schemes are PKCS#1 v1.5 and OAEP [17], both standardized in RFC 8017 [18]. We briefly recall the PKCS#1 v1.5 padding scheme, used in TLS RSA ciphersuites, which is vulnerable to Bleichenbacher’s attack.

Given a κ -byte RSA modulus and a payload message M , the padded message m has the form

$$0x00 \parallel 0x02 \parallel PS \parallel 0x00 \parallel M,$$

where PS is a string of random non-zero bytes of length at least 8. Throughout this paper, M denotes the unpadded payload and m the corresponding padded RSA plaintext. The leading (i.e., most significant) $0x00\ 02$ bytes ensure that a correctly padded message m satisfies $2B \leq m < 3B$, where $B = 2^{8(\kappa-2)}$ is a global constant of the padding scheme.

2.2 Bleichenbacher’s Attack

The Bleichenbacher attack [1], first published in 1998, allows an attacker with access to a particular *oracle* from the server holding the secret key to decrypt RSA ciphertexts. More concretely, it is an adaptive chosen-ciphertext attack against systems that use RSA with PKCS#1 v1.5 padding and expose a *padding oracle* that reveals whether a given ciphertext decrypts to a correctly padded message.

The high-level overview of the attack is simple. Suppose we want to decrypt a given ciphertext c , i.e., compute $m = c^d \bmod N$; the attacker exploits RSA’s malleability and computes $\tilde{c} = c \cdot s^e \bmod N$ for some s . This ciphertext decrypts to $\tilde{m} = m \cdot s \bmod N$. The attack then iterates over many choices of s , and for each value the padding oracle tells the attacker whether $\tilde{m}$ is PKCS-conforming. If it is, this implies that $2B \leq m \cdot s \bmod N < 3B$. By collecting many small pieces of information of this form, the attacker can derive the message m ; the original attack required around 2^{20} such queries and thus became known as the “million-message attack”.

Specifically, Bleichenbacher’s original algorithm proceeds in several phases, keeping a list of intervals whose union must contain the message m . *Step 1* initializes the search by finding (or starting from, if c already encrypts a correctly padded message) a ciphertext c_0 such that the decrypted message m_0 lies in the interval $[2B, 3B - 1]$. *Step 2a* searches for the smallest multiplier s_1 that produces a new positive oracle response, giving a first interval reduction. *Step 2b/2c* iteratively refines the interval: Step 2b handles cases with multiple disjoint sub-intervals, while Step 2c works on a single interval by computing the next multiplier s_{i+1} from the current bounds, roughly halving the interval each time. The attack repeats Step 2 until only one interval remains, containing a single value.

Capol’s comprehensive 2021 study reports that the original attack required a median of 50 191 [13, Table 3.11] oracle calls for a valid PKCS#1 v1.5 ciphertext encrypted under a 2048-bit key. This approach starts from the interval $[2B, 3B - 1]$ to which the padded message m must belong and then iteratively shrinks it. Operationally, this means that the standard attack recovers information about m from the most significant bits downward: each execution of Step 2c approximately fixes one additional high-order bit. This order is wasteful in settings where only the payload M is needed: the attack first recovers the random padding PS and only later reaches the least significant bits containing M .

2.3 Padding Oracle Types and Attack Efficiency

The attack’s efficiency depends on the “strength” of the padding oracle. Bardou et al. [6] introduced a detailed notation that characterizes oracles by the checks they perform or skip on the decrypted plaintext. The oracle types are defined by three boolean values, where ‘F’ (false) indicates that a test is conducted, and ‘T’ (true) indicates that it is skipped. The three positions correspond to checks for a zero byte occurring after the first ten bytes, for zero bytes inside the padding string, and for the encoded length.

The TTT oracle is the most permissive type and only verifies that the first two bytes of the plaintext are $0x00 \parallel 0x02$. It is therefore generally the easiest to attack. The TFT oracle checks the starting $0x00 \parallel 0x02$ bytes and the presence of at least eight non-zero padding bytes. The FFT oracle represents the standard PKCS oracle. It returns ‘true’ for correctly padded plaintexts of any length by checking $0x00 \parallel 0x02$, non-zero padding bytes, and the presence of a $0x00$ byte after the padding.

We call *false negative* an oracle query whose decrypted value lies in the $[2B, 3B)$ interval, but the oracle gives a negative answer due to the additional checks. The false-negative rate depends on the oracle type and key size: TTT oracles have no false negatives, TFT oracles have a rate of 3.1%, and FFT oracles have rates of 64.2% (1024-bit) and 40.1% (2048-bit). Equivalently, one positive response requires an expected 1.032, 2.790, and 1.669 queries for TFT, 1024-bit FFT, and 2048-bit FFT oracles, respectively. The lower rate for 2048-bit FFT keys reflects the reduced probability of a missing zero byte in a longer message.

The three types are not equally likely to occur in practice. FFT performs exactly the checks PKCS#1 v1.5 mandates, so it is the oracle exposed by an otherwise *conforming* unpadding routine whose outcome becomes observable, for example through a distinguishable error or timing behavior. TTT and TFT additionally require the implementation to *omit* mandated checks, which makes them substantially easier to attack.

Ronen et al. [12] find FFT (their FFTT) in five of nine classified TLS implementations, more than any other type. Both the vulnerable Erlang server of our real-world experimental setup (Section 6.1), and the Alibaba Cloud load balancer we examine (Appendix D) are likewise FFT oracles.

Ronen et al. [12] further extended oracle classification by considering TLS libraries that also check whether the first two bytes of the *payload* message match the TLS version. Their strictest FFFF oracle, also known as the “Bad Version Oracle” (BVO) [19], requires tens of millions of oracle queries.

2.4 Prior Bleichenbacher Optimizations

Over the last two and a half decades, researchers have developed various optimizations to reduce the algorithm’s query complexity.

Klima, Pokorný, and Rosa (KPR) [19] introduced the *Parallel Threads Method*, which improves Step 2b by searching multiple intervals concurrently rather than sequentially.

Bardou et al. [6] introduced the concept of *trimmers* and the *Skipping Holes* technique. Trimmers are small divisors of the plaintext message m that can be checked quickly; when they exist, they immediately provide a noticeable shrinking of the starting interval. The Skipping Holes technique instead provides an optimized way to search for s_1 in Step 2a, eliminating queries that would have returned a negative answer.

Capol [13] added a Step 2c heuristic that detects failing patterns and aborts unpromising attack instances early. With these optimizations, the median number of oracle calls for a 2048-bit RSA key dropped from 50 191 to 11 428 [13].

2.5 TLS and RSA Key Exchange

In TLS 1.0–1.2, the `TLS_RSA` ciphersuites use RSA encryption for key exchange [8]. During the handshake, the client generates a 48-byte PreMasterSecret consisting of a 2-byte TLS version identifier followed by 46 random bytes. The client encrypts the PreMasterSecret under the server’s RSA public key using PKCS#1 v1.5 padding and sends it in the `ClientKeyExchange` message. The server decrypts the ciphertext to obtain the PreMasterSecret, from which both parties derive the session keys. For TLS 1.2, the padded plaintext is:

$$0x00 \parallel 0x02 \parallel PS \parallel 0x00 \parallel 0x03 \parallel 0x03 \parallel S,$$

where PS is random padding. The PreMasterSecret is represented by the lowest 48 bytes, consisting of the version identifier `0x0303` and 46 random bytes S .

For this attack, the attacker only needs to recover the random bytes S of the PreMasterSecret, i.e., the least significant 46 bytes of the padded RSA plaintext. Thus, the PreMasterSecret is recovered last by the standard (MSB-first) Bleichenbacher attack, which requires first decrypting the entire padded message, including all padding bytes. This makes TLS a natural setting for recovering the least significant bytes first, since recovering the random PreMasterSecret bytes is sufficient to derive the session keys.

3 Threat Model and Attack Scenario

In general, our threat model is the same as for standard Bleichenbacher attacks: we assume there is a party that exposes a padding oracle, as in Section 2.3, with which the attacker can interact. The further details then depend on the specifics of the used protocol.

For our case study, we focus on TLS, and hence we instantiate the TLS-relevant threat model and assumptions next, but we note that similar observations can be made for, e.g., Kerberos. For TLS, there are two relevant target scenarios: (i) attacking a session between a client and server that would not negotiate RSA mode, and (ii) attacking a session between a client and server that honestly negotiated RSA encryption as their ciphersuite.

Scenario 1: Man-in-the-middle attack on parties that support RSA. Modern servers and web browsers support TLS 1.3 [20], which no longer supports RSA key exchange. However, attacking TLS using Bleichenbacher only requires both client and server to *support* RSA: it also works if the client and server additionally support stronger ciphersuites (e.g., if both support TLS 1.2 and TLS 1.3), which is often true in practice for backward compatibility. This is the scenario that we consider in this paper.

To impersonate the server, the attacker intercepts the `ClientHello` and responds with the server’s real certificate while selecting RSA. The client encrypts a fresh PreMasterSecret under the

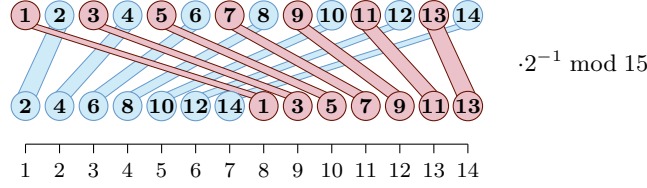


Figure 1: Multiplying the residues modulo 15 by 2^{-1} permutes them so that even values (blue) land in $[1, 7]$ and odd values (red) in $[8, 14]$, partitioning by least-significant bit.

server’s public key and waits for a valid response or a timeout. The attacker uses this ciphertext in padding-oracle queries; if the client times out, the attempt fails and a new connection uses a new secret. If the attacker recovers the PreMasterSecret in time, they derive the session keys, complete the handshake, and impersonate the server within an apparently normal TLS connection.

Notice that TLS 1.3 downgrade cookies do not prevent our attack: the attacker directly impersonates the vulnerable server with a TLS 1.2 ServerHello selecting `TLS_RSA`. The client encrypts the PreMasterSecret under the static RSA key in the server’s certificate; the attacker recovers it through the padding oracle and completes the handshake.

Scenario 2: After-the-fact decryption of honestly negotiated RSA mode. If a client and server honestly negotiate `TLS_RSA`, then an attacker can store their encrypted messages, and at any later time use the oracle (without any limitations on the number of queries) to recover the associated PreMasterSecrets and decrypt the previously stored sessions at any time. However, most modern browsers and servers support TLS 1.3, and thus would never honestly negotiate `TLS_RSA`. Hence, we do not consider Scenario 2 in the remainder of this paper, and focus solely on Scenario 1.

4 Core Algorithmic Improvements

The core idea of our new algorithm is that it is possible to recover the least significant bits of the padded message first, thereby quickly revealing the actual payload M encrypted in the RSA ciphertext without wasting queries on the random padding.

We first give a simplified intuition (visualized in Figure 1) for how our algorithm recovers the least significant bit of the padded message m , i.e., whether it is even or odd. Picture the residues $0, 1, \dots, N-1$ laid out on a number line from 0 (left) to $N-1$ (right). Now, multiplying every element by $2^{-1} \bmod N$ rearranges the residues so that even values cluster in the left half (below $N/2$) and odd values cluster in the right half (above $N/2$). In this sense, the division acts like a bit rotation: the least significant bit becomes the most significant one, and using other mathematical tricks can be tested with the oracle as in the classical Bleichenbacher attack, since the padding check considers the highest bits.

More precisely, starting from an interval $[a, b]$ containing m , the bottom-up step guesses the parity of m . After multiplication by $2^{-1} \bmod N$, the interval splits into two halves corresponding to the two possible parities, and the attack tries to align the half corresponding to the guess with the oracle range. A positive oracle response means that the guess was correct, thereby revealing the least significant bit of m . This alignment is possible only when the interval is sufficiently small, which is why we first run a preprocessing phase that narrows the interval accordingly.

More generally, multiplication by $D^{-1} \bmod N$ splits the residues into D contiguous clusters based on their residue modulo D . By choosing the multiplier carefully, we can shift a chosen residue class into the oracle range. When the candidate classes satisfy the separation conditions proved below, this tests whether $m \equiv v \pmod{D}$ with a single query, for an admissible guess $v \in \mathbb{Z}_D^*$. Iterating with $D = 2, 4, 8, \dots$ recovers the plaintext bit by bit, starting from the least significant bit.

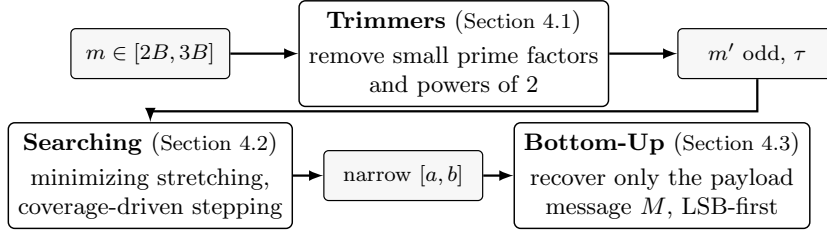


Figure 2: Overview of our algorithmic improvements. Trimmers divide out small factors, and produce a divisor τ such that $m' = m/\tau$ is odd. Searching uses τ for coverage-driven stepping to narrow the interval. Bottom-Up recovers the n least significant bits of m' .

Throughout this section, we consider the following oracle $\mathcal{O}_x$, where x denotes a generic working message:

$$\mathcal{O}_x(s) = [2B \leq (x \cdot s) \bmod N < 3B]. \quad (1)$$

This is the TTT oracle from the literature. We first state and analyze our theorems for this oracle, and then discuss in Section 4.5 how the analysis translates to other oracle types.

Our main result can be stated informally as follows.

Theorem 1 (Informal). *Let a and b be known integers with $0 < a < b < N$. Let $x \in [a, b]$ be an unknown odd integer, and let n be a positive integer. Assume access to the oracle $\mathcal{O}_x$ from Equation (1). A singleton interval already determines x without queries. If*

$$2^n + 2b \leq B/2 \quad \text{and} \quad \frac{b}{b-a} \geq \frac{N}{B},$$

then we can recover $x \bmod 2^n$ with at most $2n$ queries.

The rest of this section presents the full algorithm in the Bleichenbacher setting: first the preprocessing steps Trimmers and Searching, and then the Bottom-Up procedure itself, together with the proof of the theorem stated above. We briefly outline these three steps (visualized in Figure 2) before discussing each in more detail.

Trimmers (Section 4.1). The first condition for the theorem requires the upper bound b of the interval to be substantially below B ; thus, applying the theorem with the naive interval $[2B, 3B - 1]$ containing the padded message m would not work. The main goal of the trimming step is to find a *divisor* τ of m and start working with the smaller message $m' = m/\tau$. In this way, we can start from the interval $m' \in [2B/\tau, 3B/\tau]$: already $\tau \geq 6$ makes the upper bound smaller than $B/2$, and Section 4.4 gives the exact threshold needed. This step also divides out all factors of 2 from m and incorporates them into τ , so that m' is odd. The last substep of this phase further reduces the size of the interval.

Searching (Section 4.2). Even after trimming, the interval is often still too wide for our theorem to apply: each residue class would span more than the oracle range. The basic bit-by-bit theorem above asks for $\frac{b}{b-a} \geq N/B \approx 2^{16}$. The implementation instead makes a larger first Bottom-Up jump, which relaxes this requirement to $N/(4B)$; we formalize this variant in Theorem 3. Here, we follow a strategy analogous to Step 2a of the original algorithm, which allows us to find a smaller sub-interval to which the message belongs. The main presentation uses coverage-driven stepping with the standard target range. An optional halving optimization that further reduces the search cost is analyzed in Appendix G.

Bottom-Up (Section 4.3). When preprocessing establishes the conditions of Theorem 1, the Bottom-Up procedure computes the least significant n bits of m' in at most $2n$ queries via iterated multiplication by 2^{-i} and next-bit guessing. More generally, Theorem 3 allows a larger first jump followed by one or more bits per round. Finally, we recover the payload message M as $m' \cdot \tau \bmod 2^n$.

4.1 Trimmers

In most cases, we cannot apply Theorem 1 directly to the padded message m . Instead, we first need to find a sufficiently large divisor τ of m and then work with the reduced message

$$m' = m/\tau.$$

At the end of this phase, we obtain three outputs needed later: the accumulated divisor τ , an odd working message m' , and a smaller interval containing m' . Following Bardou et al. [6], we use *trimmer* queries to test small candidate divisors. For a candidate $t < 2N/9B$, we query multipliers of the form

$$s_{t,\varepsilon} = (t \pm \varepsilon) \cdot t^{-1} \bmod N$$

for small offsets ε . If t divides m , then the decrypted value is

$$(m \cdot s_{t,\varepsilon}) \bmod N = m \cdot \frac{t \pm \varepsilon}{t} = m \pm \varepsilon \cdot \frac{m}{t},$$

where $\frac{m}{t}$ is now an integer in the range $[2B/t, 3B/t]$. If m is not too close to the boundary of the interval $[2B, 3B - 1]$, then for a small offset ε it is plausible that at least one such query remains inside the oracle interval, i.e. that $\mathcal{O}_m(s_{t,\varepsilon}) = 1$. If t instead does not divide m , multiplication by $t^{-1} \bmod N$ no longer corresponds to integer division, and the value $m \cdot t^{-1} \bmod N$ heuristically behaves like an unrelated residue modulo N . In that case, falling inside the oracle range $[2B, 3B]$ is correspondingly unlikely.

Whenever a candidate divisor is accepted, we add it to the accumulated divisor by setting τ to the least common multiple of all accepted divisors. We then extend τ by powers of two: as long as the trimmer test indicates that 2τ still divides m , we replace τ by 2τ .

Once τ has been determined, we work with the *reduced message* $m' = m/\tau$; the last step ensures that m' is odd.

Moreover, queries to the original oracle can now be interpreted as queries about m' : calling $\mathcal{O}_m$ on a multiplier of the form $s \cdot \tau^{-1} \bmod N$ reveals whether $(m' \cdot s) \bmod N$ lies in $[2B, 3B]$. Thus, although we still interact only with $\mathcal{O}_m$, the remaining steps can be understood as operating on the reduced message m' through the induced oracle $\mathcal{O}_{m'}$.

The trimming phase also tightens the interval. In particular, we binary search for the smallest and largest accepting integers $\frac{2}{3}\tau \leq u_l \leq u_h \leq \frac{3}{2}\tau$ such that $\mathcal{O}_m(u/\tau) = 1$. Each accepting multiplier gives $2B \leq um' < 3B$. This condition holds over the integers, not only modulo N : since $u \leq 3\tau/2$ and $m < 3B$, we have $um' = u \cdot (m/\tau) < 9B/2 < N$ for PKCS#1 v1.5 parameters.

The accepting range $[u_l, u_h]$ thus yields the interval

$$\frac{2B}{u_l} \leq m' < \frac{3B}{u_h}.$$

The next phases take this interval and τ as their starting point, and can be thought of querying the induced oracle $\mathcal{O}_{m'}$.

This is also the most probabilistic part of the attack, thus possibly leading to later failures; in Section 6.3 we describe and evaluate in detail all the possible failure modes.

4.2 Searching

Searching is the analogue of Step 2a of the original Bleichenbacher attack, now applied to the reduced working message m' . Its objective is the same: starting from a current interval $m' \in [a, b]$, find a multiplier s such that querying the original oracle with $s \cdot \tau^{-1} \bmod N$ is accepted, or equivalently, such that $\mathcal{O}_{m'}(s) = 1$. A positive answer immediately yields a smaller interval for m' .

If the target oracle interval is $[L, U)$ with width $U - L = B$ and $\mathcal{O}_{m'}(s) = 1$, then there exists an integer w such that $m' \cdot s \in [w \cdot N + L, w \cdot N + U)$, and therefore

$$m' \in [a, b] \cap \left[\frac{w \cdot N + L}{s}, \frac{w \cdot N + U}{s} \right).$$

One positive query thus replaces the current interval by a shorter one of width about B/s .

We briefly describe here a simple *coverage-driven* variant of Step 2a that is convenient to analyze. A full description of the optimized variant is given in Appendix G.

Fine-grained τ -stepping. First of all, notice that the knowledge of τ lets us ask more fine-grained queries to the oracle $\mathcal{O}_m$. Step 2a’s classical search increments the multiplier by 1 at each query. Starting from the hypothesis that $m \in [2B, 3B)$, increasing the multiplier by 1 shifts the queried interval by at least $2B$. This distance is wider than the oracle range B , which misses slices.

We notice that we can use the value of τ recovered from the trimming step to perform a finer-grained search. In particular, our reduced message $m' = m/\tau$ lies in a smaller interval $[a, b]$, with $b \leq 3B/\tau$. This means that integer steps with respect to the reduced message m' shift the queried interval by around $3B/\tau$ at a time, which is less than the width B of the oracle range from $\tau \geq 3$. Thus, querying the oracle with values $s, s + \tau^{-1}, s + 2 \cdot \tau^{-1}, \dots$ restores the granularity needed to land inside that range.

Coverage-driven selection. The previous fine-grained stepping strategy, if used naively, would lead to multiple queries testing the same slice of the interval. Our solution is to pre-compute the *coverage* of a given multiplier, avoiding testing multipliers whose coverage intersect. Concretely, for integer endpoints we partition the working interval $[a, b]$ into consecutive integer slices $[a_1, b_1], \dots, [a_J, b_J]$ with $b_1 = b$ and $b_{j+1} = a_j - 1$. For each slice, we pick a multiplier s_j such that

$$N + 2B \leq s_j \cdot x < N + 3B$$

for every $x \in [a_j, b_j]$. We build the slices greedily from top to bottom: given b_j , we choose the largest multiplier that still maps b_j below the top of the oracle range, in particular

$$s_j = \left\lfloor \frac{N + 3B - 1}{b_j} \right\rfloor.$$

This current slice extends down to $a_j = \max(a, \lceil (N + 2B)/s_j \rceil)$. If $b \leq B$, as in the sufficient trimming regime below, the chosen multiplier accepts b_j and hence every slice is nonempty.

For each of these multipliers, we query the oracle on $s_j \cdot \tau^{-1} \bmod N$, equivalently testing whether $\mathcal{O}_{m'}(s_j) = 1$. A positive answer on s_j then identifies $[a_j, b_j]$ as the new interval for m' . This is the interval handed to Bottom-Up.

Query count. For this simple variant, relevant multipliers near a point x satisfy $s_j \approx N/x$, so the slice covered by s_j has width about $B/s_j \approx x \cdot B/N$. The slice widths therefore scale linearly with position: each slice has $a_j \approx b_j \cdot (1 - B/N)$, thus after J slices the lower endpoint is

$$a_J \approx b \left(1 - \frac{B}{N}\right)^J \approx b \cdot e^{-BJ/N}.$$

Reaching a thus requires

$$Q_{\text{search}} \approx \frac{N}{B} \log \frac{b}{a} \tag{2}$$

queries. Its width is at most B/s_j . Since $s_j > (N + 3B - 1 - b_j)/b_j$, the width is at most $b \cdot B/N$ when $b \leq 3B - 1$.

The optional halving variant, including its query estimate, exact rounded width bound, and integration with Bottom-Up, is analyzed in Appendices G and G.3.

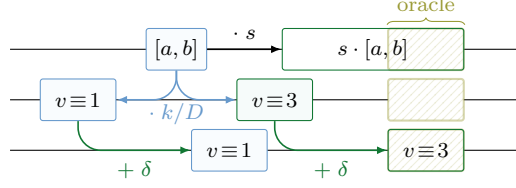


Figure 3: Bottom-Up step. Top: classical approach, recovers MSBs first. Middle, bottom: our approach, recovers LSBs first. Multiplication by k/D splits the interval into modulo groups. Offset by $+\delta$ moves the target group into the oracle range.

4.3 Bottom-Up Decryption

The core innovation of our approach is a Bottom-Up decryption procedure that recovers the least significant bits of the reduced working message first. Let x denote an unknown message, which is instantiated with the working message m' after Trimmers and Searching. Throughout this section, we consider the following negated oracle

$$\mathcal{O}_x^-(s) = [N - 3B < (x \cdot s) \bmod N \leq N - 2B],$$

which can be implemented via $\mathcal{O}_x^-(s) = \mathcal{O}_x(-s)$. We use this oracle because it allows us to reuse the optimizations described in Appendix G.

The key primitive is that, once the current interval $[a, b]$ containing x is narrow enough and the candidate residues are sufficiently separated, a single oracle query can test whether x lies in a chosen residue class modulo D . We visualize this compared to the classical approach in Figure 3. In the general statement below, the relevant guesses are the units modulo D ; in the powers-of-two application used later, x is odd, so these are the odd residues. For real t , write

$$\|t\|_N = \min_{z \in \mathbb{Z}} |t - zN|.$$

The circular distance modulo N between two real numbers u and v is $\|u - v\|_N$. For an integer t , this equals $\min(t \bmod N, (-t) \bmod N)$.

Theorem 2. *Suppose that $0 < a < b$, $x \in [a, b]$ is an integer, and $D \geq 2$ is an integer with $\gcd(D, N) = \gcd(x, D) = 1$. Suppose that*

$$\frac{2B + D + 2b}{D - 1} \leq \frac{N}{D} \leq \frac{b}{b - a} \cdot (B - D - 2b).$$

Then, for any $v \in \mathbb{Z}_D^*$, we can construct a single oracle query to $\mathcal{O}_x^-$, using a multiplier $s(v) = \delta + k \cdot \hat{D}$, with the following guarantees.

If we let $v^* = x \bmod D$ denote the correct residue class of x , we have that $\mathcal{O}_x^-(s(v^*)) = 1$, and that $\mathcal{O}_x^-(s(v)) = 0$ if $\|k(v - v^*)\hat{D}\|_N \geq B$, where $k = k(v)$ is the coefficient for the queried guess.

Thus the correct guess always passes a perfect oracle. A passing query identifies the residue only when every other possible residue satisfies the rejection condition; the batched powers-of-two proof below establishes rejection directly for its candidate set.

In the rest of the section, we describe how to construct the multiplier $s(v)$, and then how to specialize the theorem to recover the least significant bits of the unknown working message x .

The core idea is that multiplying by $D^{-1} \bmod N$ maps the elements of $[a, b]$ congruent to v modulo D into a contiguous interval. Moreover, a further multiplication by k moves the chosen residue cluster near $\frac{rN}{D}$, and in particular near the negated oracle range $(N - 3B, N - 2B]$. It permutes the residue-class indices only when $\gcd(k, D) = 1$, as ensured by odd r in the

powers-of-two application. Finally, we show that we can fine-tune our queries and compute an offset δ that moves the interval completely inside the oracle range.

A full proof of Theorem 2 is given in Appendix H, and in particular consists of the union of Lemma 4 and Lemma 5.

4.3.1 Testing One Residue Class

We now explain the shape of the multiplier $s(v)$ from Theorem 2 and the concrete parameter choices used in the implementation. In the appendix, the theorem is proved by showing that, under the stated interval bounds, there exists a valid pair (r, δ) ; here we sketch the same construction and record the empirical choices used in practice. In particular, we are given the modulus D and a guess $v \in \mathbb{Z}_D^*$. Write $v^* = x \bmod D$, so $x = D \cdot l + v^*$ for some integer l .

Choosing r . Geometrically, we want the cluster corresponding to the guessed residue class to land near the oracle range $(N - 3B, N - 2B]$. In particular, we aim for $R = rN/D \approx N - 2.5B$, i.e., the midpoint of the range. This results in the concrete choice

$$r = D - \left\lceil \frac{5BD}{2N} \right\rceil, \quad (3)$$

with an even result decremented by one for powers-of-two recovery, so that r is odd. The implementation then searches smaller odd values if needed. For the formal guarantee, we choose an admissible odd r from the strengthened part of Lemma 5; this need not be the heuristic starting value.

Notice that larger values of D give us more control on where to position the cluster of the guessed residue class. In comparison, smaller values of D require a bigger contribution from the δ offset to construct a valid multiplier.

This empirical construction of r is shown to work experimentally in Section 6. At the same time, the appendix proves the weaker formal statement that, under the hypotheses of Theorem 2, there exists *some* valid choice of r and δ , and proves the existence of an odd r under the stronger bounds used for batched recovery.

Computing k . Let $\hat{D} = D^{-1} \bmod N$ be the modular inverse of D , and define the quotient $h = \lfloor \hat{D} \cdot D/N \rfloor$ so that $D \cdot \hat{D} = 1 + hN$. The hypothesis $\gcd(D, N) = 1$ ensures that $\hat{D}$ exists. Moreover, $hN \equiv -1 \pmod{D}$ implies $\gcd(h, D) = 1$, so hv is invertible modulo D for every $v \in \mathbb{Z}_D^*$. We choose an integer k so that multiplication by $k\hat{D}$ places the guessed residue class v near the target offset R . In particular, given the choice of r from the step above and a guess of residue class v , we compute

$$k = r \cdot (h \cdot v)^{-1} \bmod D. \quad (4)$$

This ensures that $k \cdot v \cdot h \equiv r \pmod{D}$, and therefore that $k \cdot v \cdot \hat{D} \bmod N$ lands within distance at most D of R . In other words, multiplication by $k\hat{D}$ selects the guessed residue class and moves its cluster near the desired offset.

Finding δ . Once r and k are fixed, we look for an offset δ that places the interval fully within the oracle range. Given the working hypothesis that $x \in [a, b]$, the values

$$\begin{aligned} \delta_{\min} &= \max\left(0, \left\lfloor \frac{N - 3B - R}{a} \right\rfloor + 1\right), \\ \delta_{\max} &= \left\lfloor \frac{N - 2B - D - R}{b} \right\rfloor - 1. \end{aligned}$$

describe the interval of non-negative integer δ values for which

$$[\delta a + R, (\delta + 1)b + R + D] \subset (N - 3B, N - 2B].$$

For this guaranteed construction, a centered choice is

$$\delta = \left\lfloor \frac{\delta_{\min} + \delta_{\max}}{2} \right\rfloor.$$

The implementation likewise starts near the center of its candidate offset range and explores alternatives, but its range allows partial coverage of $[a, b]$ and need not coincide with the interval above.

Lemma 5 instead proves that under the hypotheses of Theorem 2 there exists an admissible choice of r for which the corresponding integer interval $[\delta_{\min}, \delta_{\max}] \cap \mathbb{Z}$ is non-empty.

Assembling and testing the query. The final multiplier is then

$$s(v) = \delta + k \cdot \hat{D}. \quad (5)$$

When the guess is correct, i.e., $v = v^* = x \bmod D$, the product satisfies

$$x \cdot s(v^*) \bmod N \in [\delta a + R, (\delta + 1)b + R + D].$$

By our choices, this interval entirely lies inside $(N - 3B, N - 2B]$, so the oracle $\mathcal{O}_x^-$ returns 1 for the correct guess. A proof of this fact is in Lemma 4.

For a common choice of r and δ , the actual correct-guess and incorrect-guess queries differ by

$$xs(v) - xs(v^*) \equiv x(k(v) - k(v^*))\hat{D} \pmod{N}.$$

To prove rejection, we instead use the auxiliary point

$$A_v = \delta x + kl + \frac{kv + rN}{D}, \quad k = k(v),$$

where $l = (x - v^*)/D$. As shown in Lemma 4, A_v is an integer inside the oracle range and

$$xs(v) \equiv A_v + k(v^* - v)\hat{D} \pmod{N}.$$

Consequently, $\|k(v - v^*)\hat{D}\|_N \geq B$ suffices to reject this incorrect guess.

4.3.2 Recovering Bits in Batches

We now specialize Theorem 2 to powers of two. Since Trimmers absorbs all factors of 2 into τ , the working message x is odd, so we initially know $x \bmod 2 = 1$. We first make a larger jump to an initial modulus $D_0 = 2^{d_0}$ by testing all $D_0/2$ odd residues. Afterward, each round multiplies the current modulus by 2^ℓ and tests the 2^ℓ compatible residue classes. The queries within a round are independent and can be issued in parallel.

Formally, suppose that we already know $X \equiv x \pmod{D_{\text{prev}}}$. For $D = 2^\ell D_{\text{prev}}$, the next round tests

$$X + zD_{\text{prev}}, \quad z = 0, \dots, 2^\ell - 1.$$

We choose odd r using the strengthened part of Lemma 5, and apply the test from Theorem 2 to every candidate. Exactly one is correct and passes a perfect oracle. The theorem below also includes the initial jump, whose branching factor is $D_0/2$ rather than 2^ℓ .

Theorem 3. *Let n, d_0, ℓ be positive integers with $2 \leq d_0 \leq n$, let $D_0 = 2^{d_0}$, and let*

$$\rho = \left\lceil \frac{n - d_0}{\ell} \right\rceil, \quad D_{\max} = 2^{d_0 + \ell\rho}.$$

Suppose that $0 < a < b$, $x \in [a, b]$, and x is odd. If

$$D_{\max} + 2b \leq B/2, \quad \max\left(\frac{D_0}{2}, 2^\ell\right) \leq \frac{N}{B}, \quad \text{and} \quad \frac{b}{b-a} \geq \frac{4N}{D_0B},$$

An extended abstract of this paper is in the proceedings of NDSS 2027. This is the full version.

then we can first recover $x \bmod D_0$ by testing the $D_0/2$ odd residues modulo D_0 , and subsequently recover ℓ additional bits per round. This recovers $x \bmod 2^n$ using at most

$$\frac{D_0}{2} + \rho 2^\ell$$

oracle queries in $1 + \rho$ adaptive rounds.

We prove Theorem 3 in Appendix H.3.

The sequential implementation takes $D_0 = 16$ and $\ell = 1$. The required width is then $b/(b-a) \geq N/(4B)$ and the query bound is $8 + 2(n-4) = 2n$. Uniform ℓ -bit batching from the known parity is the special case $d_0 = 1 + \ell$: it uses $\lceil (n-1)/\ell \rceil$ rounds and at most $\lceil (n-1)/\ell \rceil 2^\ell$ queries. Separating d_0 from ℓ also allows a modulo-16 kickstart before switching to larger parallel batches.

4.4 The Full Algorithm

We can now summarize the full attack. Let τ be the divisor returned by Trimmers and let $m' = m/\tau$. Since τ divides m and all powers of 2 have been absorbed into τ , the working message m' is odd and satisfies the basic trimming interval

$$m' \in \left[\frac{2B}{\tau}, \frac{3B}{\tau} \right).$$

The final binary search inside Trimmers then finds accepting integers $2/3\tau \leq u_l \leq \tau \leq u_h \leq 3/2\tau$ with $\mathcal{O}_{m'}(u) = 1$, which refines the interval to

$$m' \in [a', b'] := \left[\left\lceil \frac{2B}{u_l} \right\rceil, \left\lfloor \frac{3B-1}{u_h} \right\rfloor \right].$$

The Searching step now starts from the interval $[a', b']$ and tests one multiplier per slice. This step returns the accepting slice

$$[\tilde{a}, \tilde{b}] \subseteq [2B/u_l, 3B/u_h].$$

For standard searching, the approximation in Section 4.2 is $\tilde{b} - \tilde{a} \approx \tilde{b}B/N$, with exact width bound

$$\tilde{b} - \tilde{a} \leq \frac{b'B}{N} \leq \frac{3B^2}{Nu_h}.$$

Under the sufficient trimming condition $\tau \geq 13$ below, $b' \leq 3B/13 < B$, which also ensures that the standard-search slices are nonempty.

The attack now runs the Bottom-Up step from Theorem 3 on this narrower interval with $D_0 = 16$ and $\ell = 1$. It tests the eight odd residues modulo 16 and then adds one bit per round. If $\tilde{a} = \tilde{b}$, the message is already determined. Otherwise, we apply the theorem to $[\tilde{a}, \tilde{b}]$ and $x = m'$, implementing $\mathcal{O}_x^-(s)$ as $\mathcal{O}_m(-s/\tau)$. We check its three conditions.

Indeed, using $\tilde{b} \geq a' \geq 2B/u_l$ and the standard-search width bound we have

$$\frac{\tilde{b}}{\tilde{b} - \tilde{a}} \geq \frac{a'}{3B^2/(Nu_h)} \geq \frac{2B/u_l}{3B^2/(Nu_h)} = \frac{N}{B} \cdot \frac{2u_h}{3u_l}.$$

Since $u_h \geq u_l$, standard searching therefore gives

$$\frac{\tilde{b}}{\tilde{b} - \tilde{a}} \geq \frac{2N}{3B} > \frac{N}{4B}.$$

This establishes the width condition $\tilde{b}/(\tilde{b} - \tilde{a}) \geq N/(4B)$ required by Theorem 3 for $D_0 = 16$.

Second, the initial round has $D_0/2 = 8$ candidates. The condition $8 \leq N/B$ is immediate in the PKCS#1 v1.5 setting; in particular, $N/B \geq 2^{15}$ for the 1024- and 2048-bit moduli used in our simulations. Later rounds have only two candidates and therefore satisfy the same separation condition with additional margin.

Finally, a sufficient condition for $2^n + 2\tilde{b} \leq B/2$ is $2^n + 6B/u_h \leq B/2$, since $\tilde{b} \leq b' \leq 3B/u_h$. More precisely, the conditions

$$u_h \geq 13 \quad \text{and} \quad 2^n \leq \frac{B}{26}$$

make the size bound hold, since $B/26 + 6B/13 = B/2$. This quantifies the regime where the payload term is negligible compared to B . Since $u_h \geq \tau$, the same payload-length condition together with $\tau \geq 13$ is sufficient for the size bound. For example, recovering the 46-byte random suffix of a TLS PreMasterSecret means $n = 368$. Even for 1024-bit RSA, $26 \cdot 2^{368} < 2^{1008} = B$, so the payload-length condition holds with ample margin and $\tau \geq 13$ suffices for the size bound.

The threshold $\tau \geq 13$ is a simple uniform guarantee, not a sharp boundary. Applying the odd-offset condition of Lemma 5 directly at $D = 16$ with the standard-Searching bounds gives, up to negligible TLS-size terms, $u_h - 6 \geq 3u_l/16$. Thus $\tau \geq 7$ is often enough in practice—in particular when $u_h \geq 8$ —although it is not a uniform guarantee.

The Bottom-Up step thus correctly recovers $m' \bmod 2^n$, and the attack finally returns

$$m' \cdot \tau \bmod 2^n.$$

The total number of oracle queries is computed as

$$Q_{\text{total}} = Q_{\text{trim}} + Q_{\text{search}} + Q_{\text{BU}}.$$

The expected number of Searching queries is

$$Q_{\text{search}} \approx \frac{N}{B} \log \frac{b'}{a'} \approx \frac{N}{B} \log \frac{3u_l}{2u_h}.$$

Empirically, this number is lower with bigger values of τ (cf. Figure 12), and sometimes is 1, meaning that the binary search already shrunk the interval enough.

Regarding Bottom-Up, Theorem 3 with $D_0 = 16$, $d_0 = 4$, and $\ell = 1$ gives

$$Q_{\text{BU}} \leq 8 + 2(n - 4) = 2n.$$

The bound counts all candidates in every round. Sequentially, the correct candidate is found after about half of each candidate list, which explains the empirical estimate $Q_{\text{BU}} \approx 1.5n$ for the binary rounds.

This discussion also clarifies the tradeoff between Trimmers and Searching. Spending more queries in Trimmers can increase τ , which decreases the upper bound b' , improves the width bound $(\tilde{b} - \tilde{a}) \leq 3B^2/(Nu_h)$, and gives finer $1/\tau$ stepping in Searching. On the other hand, some plaintexts simply do not have useful small divisors in the tested range, and the smallest trimmers are the hardest to detect reliably in the presence of false negatives. We view trimming as a best-effort preprocessing phase: extra queries can help, but only up to a point, as the remaining cost depends on both the divisor found and the work needed to find it.

Consequently, with a perfect oracle, a correctly recovered divisor τ for which m/τ is odd, and offsets chosen by the proved construction, the search and Bottom-Up phases recover the requested suffix under the sufficient conditions

$$\tau \geq 13, \quad 8 \leq N/B, \quad 2^n \leq B/26,$$

for the sequential schedule with $D_0 = 16$. These sufficient conditions do not assert that Trimmers always finds such a divisor. The current optimized TTT Search preserves coverage by completing the full halving grid after sparse retries; its best-effort Bottom-Up query order is evaluated separately from the proved sequential schedule.

4.5 Handling Imperfect Oracles

We use the following strategies to handle false negatives:

Trimming. Instead of querying only $t \pm 1$, we query $t \pm \varepsilon$ for small values of ε up to a configurable bound, increasing the probability of detecting a valid trimmer.

Searching. For each candidate multiplier s_j , we also test $s_j \pm \varepsilon$ for small offsets, widening the net around each candidate.

Bottom-Up. Since δ is chosen at the center of the valid range $[\delta_{\min}, \delta_{\max}]$, we can test nearby shifts $\delta \pm \varepsilon$ for small ε while keeping r and k fixed. Each additional perturbation ε produces a different multiplier $s(v) = (\delta \pm \varepsilon) + k \cdot \hat{D}$ at negligible extra cost, increasing the probability that at least one query avoids a false negative.

5 Parallelization

When targeting a web server, the limiting factor for a Bleichenbacher attack is not computation, but network latency. The total network time is the product of the network latency L_{net} , the number of sequential queries Q_{seq} , and the number of round-trips per query R_q . For a complete handshake with $L_{\text{net}} = 5$ ms, $Q_{\text{seq}} = 1801$, and $R_q = 3$, this network time is almost 27 s. All three variables can be addressed.

Reducing L_{net} . Instead of performing the attack from the machine intercepting the victim traffic, it can be offloaded to a server geographically close to the victim server to reduce the ping.

Reducing R_q .

Normal TLS handshakes require at least three network round-trips. Our oracle queries omit the final server message, which is unnecessary once the oracle response has been observed, reducing the cost to $R_q = 2$. TCP Fast Open could reduce this to one round-trip, but server support is inconsistent, thus we do not assume it in our analysis.

Reducing Q_{seq} . Instead of sending all queries *sequentially*, independent queries can be batched and sent *in parallel*. Even if the server processes all packets sequentially, the packets can all travel through the network in parallel, reducing the network penalty. This only works for independent queries, i.e., each query does not depend on the result of any other query in the same batch. We outline for each step how requests can be batched.

Trimmers. The initial step of testing a predetermined set of possible trimmers can be fully batched. When batching queries, no prior trimmers are known, meaning $\tau = 1$. This slightly increases the chance of missing a trimmer. Additionally, all possible trimmers are batched, complicating an early abort once a suitable τ is found. Binary searching the trimmer bounds and ensuring all multiples of two are found takes less than 20 round-trips on the median, so we do not optimize it. In each step, the original implementation uses multiple queries to check a single trimmer, reducing false negatives. Once an oracle query comes back positive, no additional queries are made. Batching loses this early abort opportunity.

Searching. The searching step tests several candidates and aborts once a single one matches the oracle. These candidates are fully independent and can be batched. The early abort condition might be delayed by the batch size. We empirically identify 128 as the batch size to keep all CPU cores busy without slowing down the attack due to excessive process creation.

Bottom-up search. The final step of the attack relies on a binary search of depth n to recover the n bits of the secret. For TLS RSA key exchange, the target is the 46-byte suffix of the PreMasterSecret, so $n = 368$. Performing the full 368 round-trips would bottleneck our implementation. We thus use the uniform-batching specialization of Theorem 3: take $d_0 = 1 + \ell$, so the first and every later round check 2^ℓ independent candidates in parallel. Starting from the known parity bit, this needs $\lceil (368 - 1)/\ell \rceil$ rounds and at most $\lceil (368 - 1)/\ell \rceil 2^\ell$ queries for a perfect oracle under the theorem’s interval conditions. Choosing the optimal ℓ is now a tradeoff between local computational power and network latency. A big ℓ reduces the number of rounds,

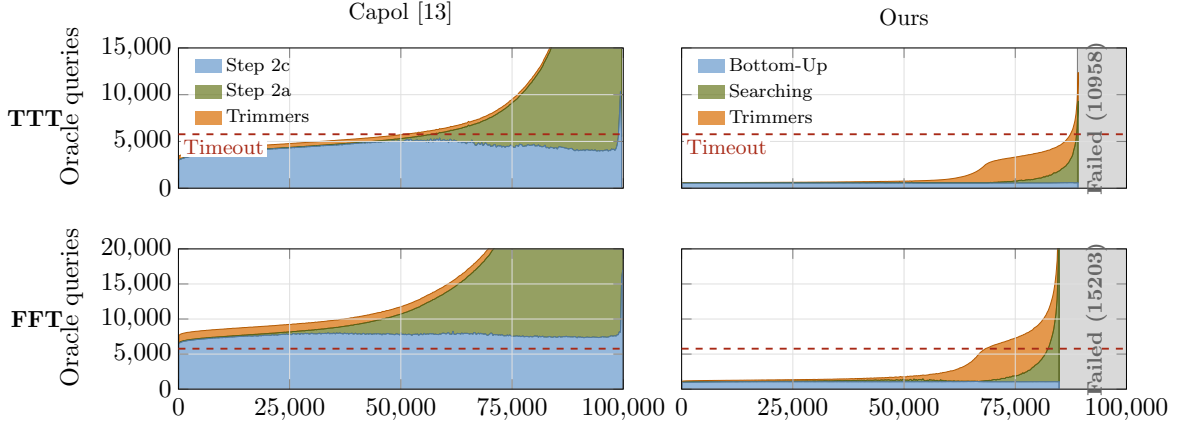


Figure 4: Per-phase query distributions for Capol and our attack with 2048-bit keys and 368-bit secrets (100 000 runs per row, grouped into 401 equal-size bins). The dashed line is the 5769-query budget from Equation (6); gray denotes algorithm failures.

but increases the total number of queries¹. In our setup and for our experiments, we choose a small $\ell = 4$.

6 Evaluation

In this section, we evaluate our optimization compared to past work. We evaluate Reverse-Bleichenbacher along three dimensions: (1) query complexity compared to prior work, (2) practical attack performance under realistic network conditions, and (3) browser-imposed constraints on online exploitation. We combine large-scale simulations with controlled real-world experiments to measure both algorithmic improvements and practical constraints.

6.1 Experimental Setup

We implement Reverse-Bleichenbacher in a framework supporting both simulation and real-world attacks. Our implementation consists of (i) the optimized attack algorithm, (ii) a simulation backend based on Capol [13] for controlled experiments, and (iii) a network attack stack that translates oracle queries into TLS handshakes using a modified `tlslite-ng` server and the ROBOT-detect tooling [3].

Simulation setup. For controlled evaluation, we simulate TTT, TFT, and FFT oracles with 1024-bit and 2048-bit RSA keys². Following prior work [13], we exclude factorable 512-bit keys. The current comparison uses 100 000 trials per configuration. Our optimized attack and the Capol baseline share master seed 20261006, the same message generator, and the same RSA key pool. Each trial index therefore uses the same message and key in both implementations. Our optimized attack uses the sparse-search retry and the TTT full-grid fallback. We report medians with failed attacks in the upper tail. The headline attack uses sparse asymmetric halving in Searching and best-effort Bottom-Up. We also run a 1000-trial ablation with standard Search coverage and the proven Bottom-Up schedule. Its message and key inputs match the first 1000 trials of the headline grid.

¹For $\ell = 1, 2, 3, 4, 5$ the queries-per-bit are 2, 2, 2.7, 4, 6.4 respectively

²We omit a 4096-bit simulation comparison because Capol’s published implementation is not optimized for this key size. However, our algorithm handles 4096-bit keys with the same asymptotic query complexity, as demonstrated in Section 7.

Real-world setup. For practical evaluation, we deploy a full man-in-the-middle attack environment. The victim server runs Erlang 20.1.6 with a known 2048-bit FFT oracle (CVE-2017-1000385), while the attacker server performs TLS downgrade and oracle querying. Our target is the 46-byte (368-bit) random suffix of the PreMasterSecret, whose full value is encrypted in the ClientKeyExchange message. Using `curl`, we run the sequential and batched implementations end to end, artificially injecting 0 ms to 100 ms latency using `tc`. We attempt 100 attacks per implementation at each evaluated RTT.

6.2 Simulation Results

We report the results of our simulations in Table 2, which shows the failure-aware median query count and both the overall and the timeout-inclusive success rate for both attacks in each configuration. We show the per-phase query distributions for the 2048-bit TTT and FFT configurations in Figure 4; Appendix E adds TFT and 1024-bit results. In the next paragraphs we evaluate the improvements in the query complexity and in the feasibility of the attacks under realistic network conditions. Section 6.3 explains algorithm failures.

Query Complexity. Across configurations, we reduce median queries by $3.97\times$ – $4.52\times$ at 1024 bits and $6.53\times$ – $7.77\times$ at 2048 bits. For FFT-2048, the median falls from 11 756 to 1801. Our Capol rerun median is within 5 % of the published FFT-2048 value of 11 428 queries. The main reason for this improvement is algorithmic and comes from reversing the recovery direction. Prior attacks recover the encoded RSA plaintext from its most significant end and must therefore recover the full 1024- or 2048-bit value before reaching the 368-bit PreMasterSecret suffix. Bottom-Up starts from the least significant end and stops once it has recovered those 368 bits, which are sufficient to derive the TLS session keys. It therefore tests fewer candidates to recover fewer bits, giving a rough bit-count advantage of $1024/368 \approx 2.78\times$ and $2048/368 \approx 5.57\times$ in the recovery phase. In the FFT-2048 median-total-query rank bin, our Bottom-Up phase uses a median of 1055 queries, whereas Capol’s analogous Step 2c uses 8033, mainly due to the recovery of the full plaintext. Our preprocessing also improves: Searching uses 331 median queries, compared with 2682 for Capol’s analogous Step 2a. Due to our reliance on finding a good trimmer, our algorithm tries to aggressively find them, explaining the abundance of trimmer queries in the tails.

Proof-backed ablation. With standard Search coverage and the proven Bottom-Up schedule, the 1000-trial ablation reaches a 2283-query FFT-2048 median and 79.2 % overall success. The optimized configuration reaches 1801 queries and 84.8 % overall success over 100 000 trials. These aggregates cover different numbers of trials, although the ablation inputs match the first 1000 optimized trials.

Simulated timeout feasibility. We also consider the feasibility of the attacks in realistic scenarios, in particular with a browser-imposed timeout, and network latency. We conducted

Table 2: Median queries and percentage of 100 000 runs that recover the 368-bit secret at any cost, and (in parentheses) within the 5769-query budget from Equation (6).

Oracle	Key Size	Capol		Ours	
		Median	Success %	Median	Success %
TTT	1024	3261	99.7 (71.8)	760	89.1 (87.5)
	2048	5714	99.7 (51.0)	759	89.0 (87.4)
TFT	1024	3675	99.7 (68.4)	813	88.5 (86.6)
	2048	6314	99.7 (39.9)	813	88.4 (86.5)
FFT	1024	16 498	99.9 (0.0)	4152	82.6 (60.3)
	2048	11 756	99.8 (0.0)	1801	84.8 (68.3)

experiments to determine realistic scenarios. Section 8.3 measures a median RTT of 2.6 ms, while Section 8.4 reports 30 s as the strictest browser timeout. Section 5 optimizes the number of round-trips per query to $R_q = 2$.

We distinguish algorithmic success from timeout feasibility: an attack is *successful within the timeout* if it recovers the target 368-bit secret within

$$5769 = \left\lfloor \frac{30 \text{ s}}{2.6 \text{ ms} \times 2} \right\rfloor \quad (6)$$

queries. We use this query budget to translate the simulated query distributions in Figure 4 into online-attack outcomes.

Capol’s algorithm recovers the secret in almost every run, but only 51.0 % of its TTT runs and none of its FFT runs stay within the budget. Ours succeeds in 89.0 % of TTT and 84.8 % of FFT runs, and in particular 87.4 % and 68.3 % of all runs stay within the query budget. The remaining 1.6 % and 16.5 %, respectively, succeed but exceed the timeout budget. Thus, while online attacks against TTT were already feasible, our query reduction technique is what makes online attacks against an FFT oracle feasible within the timeout of a single connection.

6.3 Failure Modes

We instrument every simulation to record the aborting phase and each failure type. For each target plaintext, we also derive an offline divisibility upper bound on the accumulated trimmer from the configured candidate set, $\tau_{\text{avail}} = \gcd(m, \text{lcm}(t_1, \dots, t_k))$, whether or not the attack identifies it. We now describe all the possible failures of our algorithm, quantify them in Table 3, and extensively analyze trimmers behaviour in Appendix F.

Intrinsic failures. These failures are fixed by the target plaintext and occur even with a noiseless TTT oracle. We classify a run with $\tau_{\text{avail}} \leq 2$ as a *no divisor* failure: the candidate set provides no usable accumulated trimmer³. At 2048 bits, this accounts for 9.10 % of all runs. Instead, a *no multiplier* failure means that a candidate t does divide m , but for every tested offset ε , the transformed value $m(t \pm \varepsilon)/t$ falls outside $[2B, 3B)$. Consequently, even a perfect oracle never returns the positive response needed to confirm that divisor; this accounts for another 1.84 % of 2048-bit runs. The two modes total 10.84 % at 1024 bits and 10.94 % at 2048 bits under every oracle, limiting per-ciphertext success to about 89 % with the current candidates.

Oracle-induced failures. These occur when a false negative rejects a conforming query. A *missed trimmer* means that Trimmers failed to accumulate an available factor, either because the oracle rejected a usable initial query or because the next required power of two was missed. In the latter case, the run first aborts in Searching: the observed abort step is Searching, but offline analysis attributes the cause to Trimmers. All TFT and FFT Search failures contain a TTT-conforming query that their stricter oracle rejected. No TTT run fails in Search. A Bottom-Up abort, if present, is kept unresolved until its transcript is causally classified. At 2048 bits, oracle-induced failures account for 0 %, 0.60 %, and 4.25 % of TTT, TFT, and FFT runs. For FFT-2048, the 4.25 % total comprises 3.21 % missed trimmers and 1.04 % Search false negatives, showing that Trimmers remains the principal exposure to false negatives. The larger FFT-1024 total (6.58 %) also explains why that configuration is slower and less successful than FFT-2048.

6.4 Latency and Batching Results

Figure 5 reports the real-world feasibility results. Sequential measurements cover RTTs through 30 ms, and batched measurements through 100 ms.

³A factor of 2 alone is insufficient because the confirming multiplier must be coprime to the candidate (Section 4.1)

At an intra-region 1.0 ms RTT, the sequential and batched attacks have median runtimes of 4.5 s and 2.8 s, respectively, well within browser timeouts (Section 8.4). At 20 ms RTT, the sequential median rises to 59.3 s, whereas batching reduces it to 8.3 s; latency therefore determines when batching becomes necessary.

7 Case Study: Alibaba Cloud

In this section, we complement the controlled evaluation in Section 6 by presenting an end-to-end case study. We show how a vulnerability in shared cloud infrastructure in the Alibaba Cloud can lead to the compromise of applications deployed behind it, specifically demonstrating possible remote code execution against users using the BigNox Android emulator [21].

A significant fraction of the vulnerable domains identified in our Internet-wide scan (cf. Section 8) are hosted on Alibaba Cloud. We find that this concentration is caused by shared TLS-termination infrastructure: Alibaba Cloud load balancers expose a padding oracle vulnerable to Bleichenbacher attacks. This includes a freshly provisioned load balancer using the default TLS configuration that we tested in August 2026. Consequently, connections terminated by such a vulnerable load balancer expose the oracle, *independently of the backend application or its TLS configuration*.

Prevalence. In the Tranco Top 1M, we identify 224 domains using Alibaba Cloud load balancers that expose a padding oracle. Extending our analysis with Shodan, we find over 30 000 potentially vulnerable hosts worldwide. The use of wildcard certificates further amplifies the attack surface, as a single vulnerable endpoint can be leveraged to impersonate multiple domains. We responsibly disclosed this issue to Alibaba Cloud. They acknowledged the vulnerability but had not mitigated it as of August 28, 2026.

Root Cause. Scanning for Bleichenbacher vulnerabilities does not reveal the exact TLS implementation or version. Therefore we cannot, without additional fingerprinting, identify the affected libraries.

Impact on real applications. We analyze two widely used Android emulators, BigNox [21] and LDPlayer [22], which rely on Alibaba Cloud infrastructure. Both applications implement an update mechanism that periodically queries a remote API and downloads executable updates over TLS.

Both update mechanisms rely on TLS as their only authenticity check; the downloaded executables are not independently authenticated, for example with a digital signature. Breaking the TLS connection therefore lets an attacker inject an arbitrary update payload.

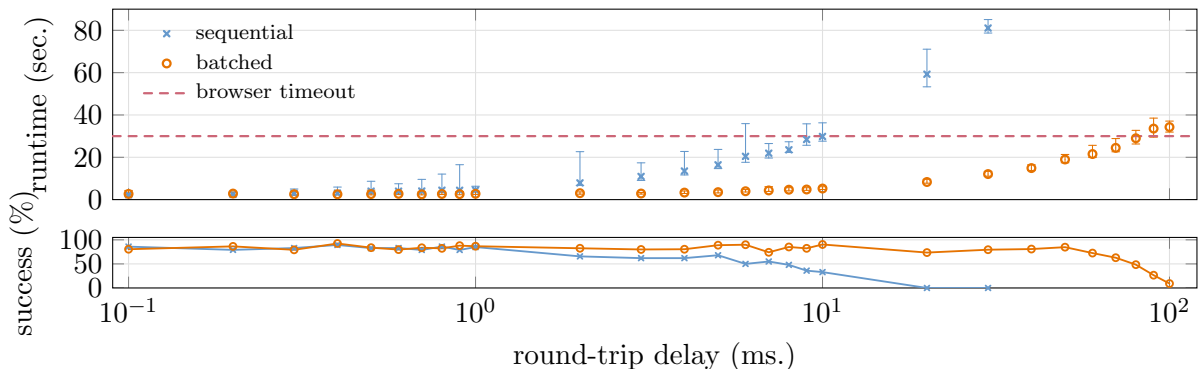


Figure 5: End-to-end performance versus RTT for sequential and batched attacks (100 attempts per RTT). Top: median and IQR of successful runtimes. Bottom: below-timeout success rate, excluding measurement errors.

End-to-end attack. We demonstrate a full MitM attack against BigNox. We assume a network attacker who can intercept and modify traffic (e.g., via ARP spoofing or control over a network path) but cannot obtain trusted certificates.

The attacker identifies relevant TLS connections using the Server Name Indication (SNI) or destination IP address. The emulator application performs regular update checks via specific API requests. The attacker selectively intercepts these requests while forwarding all other traffic unchanged. The attacker redirects intercepted requests to an attacker-controlled server in the same Alibaba Cloud region as the vulnerable load balancer, reducing network latency below 1 ms for fast oracle queries.

Using Reverse-Bleichenbacher, the attacker recovers the PreMasterSecret, derives the TLS session keys, and injects a malicious update response containing a download URL for an attacker-controlled executable.

From the user’s perspective, the application behaves as expected and prompts for an update (Figure 8). Upon confirmation, the malicious executable is downloaded and executed, resulting in remote code execution on the victim device. Notably, the load balancer uses a 4096 bit RSA key. This confirms the argument of Section 6.2: because Reverse-Bleichenbacher recovers only the 368 bit PreMasterSecret rather than the full plaintext, our cost is largely independent of the modulus size, so the improvement over prior work is even larger here than in the 2048-bit setting we benchmark.

8 Measuring the Real-World Attack Surface

In this section, we quantify Reverse-Bleichenbacher’s practical impact by measuring the prevalence of padding-oracle bugs on the Internet and checking whether the conditions for our attack are met in practice. We follow prior large-scale studies on TLS misconfigurations and oracle attacks [7] and focus on popular domains, where the impact of a successful attack is highest. We also analyze the timeout behavior of popular browsers and find an issue in Firefox’s timeout mechanisms that allows longer timeouts.

8.1 Scanning Methodology

We base our scan on the Tranco Top 1M list [15]. For each domain, we attempt a TLS handshake using ZGrab2 [23] and record the supported ciphersuites. We restrict our analysis to domains whose TLS endpoints offer TLS_RSA-based ciphersuites, as only these are susceptible to Bleichenbacher attacks.

We then test the corresponding TLS endpoint, using the domain as SNI, for padding-oracle behavior. To improve robustness, we combine two detection tools, the original ROBOT scanner [3] and SSLyze [24]. A domain is considered vulnerable if at least one tool reports oracle behavior and the result is reproducible.

8.2 Prevalence of Vulnerable Domains

Out of the 1 000 000 domains in the Tranco list, 889 384 respond to TLS handshakes. Among these, 312 289 support TLS_RSA-based ciphersuites, confirming that RSA key exchange remains widely deployed despite its removal in TLS 1.3 [20, 10, 11].

Within this set, we identify 640 domains that expose a padding oracle. This corresponds to 0.064 % of all scanned domains. While this fraction is small, it is consistent with prior work [7].

We attribute the vulnerable domains to their hosting providers using IP-range matching and ASN-based fallback. Figure 6 reports the provider distribution among the 640 vulnerable domains. For context, we compare selected provider shares with the full Tranco list and find that the vulnerable domains are far more concentrated: Alibaba Cloud accounts for 224 of the 640

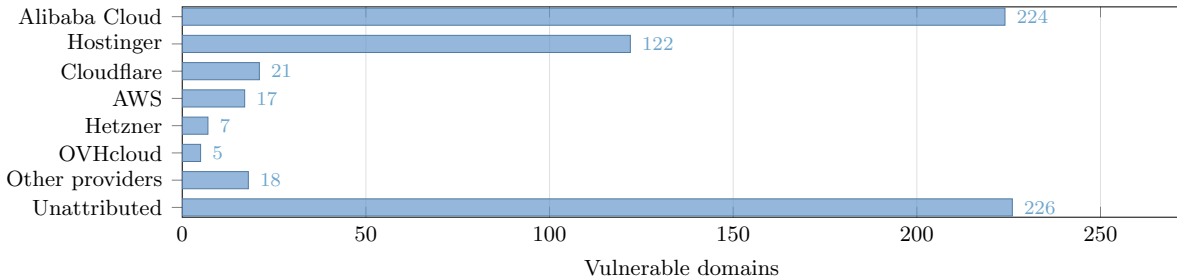


Figure 6: Hosting providers of the 640 vulnerable domains.

domains and Hostinger for another 122, together 83.6 % of the 414 we can attribute. Cloudflare and AWS, which represent 38.7 % of Tranco, account for only 21 and 17 domains.

Our attribution has two consequences. First, we identified issues stemming from cloud provider infrastructure, such as the Alibaba Cloud load balancer (cf. Section 7). Second, this shows that getting a low-latency connection is as simple as renting a machine from the victim’s cloud provider.

8.3 Latency and Attack Feasibility

The main practical constraint for Bleichenbacher attacks is network latency. We measure the minimum RTT between the attacker-controlled infrastructure and vulnerable hosts.

Using the Globalping network [25], we probe each vulnerable host from multiple locations (10 datacenter probes per continent) and record the lowest observed latency.

Across all vulnerable hosts, we observe a median RTT of 2.6 ms, and 90 % of hosts are reachable within 37 ms. Cloud deployments allow attackers to place servers in the same region as the target, reducing RTT to a few milliseconds. Combined with Reverse-Bleichenbacher’s reduced query complexity (Section 6), these latencies suffice to complete attacks within browser timeouts.

8.4 Browser Timeout and Retry Behavior

To understand the practical feasibility of downgrade attacks in browsers, we measure handshake timeouts and retry behavior. All 3 browsers tested (Chrome 146, Firefox 149, Safari 18.6) support RSA key exchange. Chrome has a hard handshake timeout of 30 s with 1 retry connection under TCP reset. Firefox’s timeout varies substantially between runs due to an implementation issue. Connection timeouts are only checked when the next connection happens; if no unrelated connections happen, this can lead to timeouts of up to 1 hour. We disclosed this to Firefox, who are discussing options such as displaying a warning (broken padlock) for RSA-based connections.

Modern browsers also open multiple concurrent TLS handshakes per origin (Firefox 2, Chrome 3) and accept the first that completes. This allows an attacker to retrieve multiple encrypted PreMasterSecret samples, making the trimmer phase of the attack more reliable. In our 300 s tests, Firefox additionally initiated up to 9 retry connections, Chrome retried 2 times, and Safari showed no retry behavior and did not timeout the handshake.

9 Discussion

Our results shift the boundary between theoretical feasibility and practical exploitability of Bleichenbacher-style attacks. The main takeaway is that residual support for RSA PKCS#1 v1.5, combined with even subtle oracle leakage, remains a practical security risk.

Implications for RSA Deployments. As long as RSA key exchange remains reachable, whether due to fallback, misconfiguration, or shared infrastructure, it provides a viable entry

point for adaptive chosen-ciphertext attacks. An active attacker can induce RSA negotiation even when endpoints do not choose it under benign conditions. Consequently, deployments that keep RSA key exchange for compatibility reasons should be treated as actively exposed rather than legacy.

Algorithmic Tradeoffs. Reverse-Bleichenbacher explicitly prioritizes median-case performance over guaranteed success. This is useful in man-in-the-middle attacks, where attackers can repeat attempts across multiple connections. Still, the empirical success rate above 82 % (60 % considering under-timeout successes) shows that this tradeoff is useful in practice. In practice, reducing *sequential* oracle queries matters more than minimizing total queries.

Network Latency. Latency constrains the attack but is not a reliable defense: co-location can achieve the low round-trip times we observe for many vulnerable hosts.

Shared Infrastructure. In practice, certificates and TLS termination are frequently shared across services, especially in cloud and CDN environments. Thus, an attacker can exploit any endpoint that exposes a padding oracle and presents a certificate valid for the target domain (including wildcard certificates). As a result, vulnerabilities in shared infrastructure can propagate across otherwise independent applications. Our Alibaba Cloud case study illustrates this amplification effect. Although the backend application does not implement the oracle, vulnerable TLS termination breaks the authenticity guarantee on which it relies.

Different Oracles. While our evaluation focuses on explicit oracle behavior, the algorithm is not tied to a specific oracle. Timing variants [4] require repeated measurements, but Reverse-Bleichenbacher reduces the adaptive rounds that are typically most expensive under noise.

Limitations. The advantages of Reverse-Bleichenbacher depend on the structure of the target plaintext. The method is most effective when only a suffix is required, as in TLS; stricter oracle models may reduce this advantage by limiting manipulation of low-order bytes.

Countermeasures. The only definitive countermeasure is to eliminate the oracle, for example by disabling `TLS_RSA` or implementing uniform, constant-time PKCS#1 v1.5 processing; Appendix C discusses these options.

10 Conclusion

We presented Reverse-Bleichenbacher, a novel variant of the Bleichenbacher attack that recovers plaintext from the least significant bits upward. By recovering only the relevant part of the message and applying improved optimizations, we reduce oracle queries enough for a substantial fraction of FFT-2048 instances to finish on one server within browser timeouts under the network conditions we measure. Our evaluation shows that these improvements translate to real-world impact. Despite the age of the original Bleichenbacher attack, padding-oracle vulnerabilities persist in deployed systems, and the low-latency conditions required for exploitation are readily achievable. These findings demonstrate that Bleichenbacher-style attacks remain a practical threat and reinforce the need to eliminate `TLS_RSA`-based ciphersuites from production systems.

Acknowledgment

The authors would like to thank Leon Trampert for his valuable assistance with scanning the real-world attack surface and setting up the pipeline for notifying affected sites. We also thank SCHUTZWERK for their support of this research.

11 Ethical Considerations

Our paper presents an optimized attack against a well-known vulnerability class that has been widely studied and exploited in the past. We structure our discussion around the affected

stakeholders, the risks and benefits of disclosure, our disclosure process, and the methodology of our Internet-wide measurements.

Stakeholder Analysis. The attack we describe affects four groups. End users face the risk that an attacker who can both intercept their TLS traffic and reach a vulnerable server may decrypt connections. Browser vendors and TLS client developers benefit from a more precise quantification of the attack. Cryptographic library and server developers can use our results to re-evaluate the urgency of patching residual oracles in their implementations. Server operators, including cloud load balancers and CDNs, can identify whether their deployments expose the oracle and choose mitigations. Customers of these services can additionally test their providers and demand fixes or switch to more secure alternatives.

Risks and Benefits of Disclosure. Successful exploitation requires two conditions: an attacker positioned to act as a man-in-the-middle on the victim’s TLS connection and a server that exposes a padding oracle. Our work does not change the difficulty of obtaining a man-in-the-middle position. What our work changes is how easily the second condition can be exploited once the first is met. We argue that publication provides attackers with little new advantage for three reasons. First, the existence of Bleichenbacher-style oracles is detectable with public tools, so motivated attackers can already enumerate vulnerable hosts. Second, prior work has already demonstrated conditions under which man-in-the-middle Bleichenbacher attacks become practical such as the multi-server attacks of Ronen et al. [12]. Third, an attacker still needs the man-in-the-middle position, on which our paper has no influence. In contrast, the defender side benefits substantially. Library developers, server operators, and browser vendors can make more informed decisions on whether to patch, deprecate, or downgrade trust in TLS_RSA ciphersuites.

Responsible Disclosure. We disclosed our findings to the affected parties. We provide a detailed timeline for the different actors: Alibaba Cloud, the Android emulators, vulnerable domains outside Alibaba networks, and browser vendors.

Alibaba Cloud On March 18, 2026, we reported the padding oracle on the Alibaba Cloud load balancer to the Alibaba Security Response Center (ASRC) under the submission *xianzhi-2026-03-193950*. ASRC responded three weeks later on April 8, 2026: they claim that they “have updated the relevant documentation to advise users against using RSA key exchange with TLS 1.2”. However, the linked documentation still recommends TLS 1.2⁴. We contacted ASRC again on April 9, April 16, and May 27, 2026, but have not received a response. As of August 6, 2026, the Alibaba Cloud load balancers remained vulnerable to Reverse-Bleichenbacher. Since the cause is at Alibaba, this is the appropriate point to report and fix, and we cannot enumerate all Alibaba users. However, after Alibaba failed to fix, we reached out to two users we know.

Android emulators On August 14, 2026, we reported to LDPlayer and BigNox that their use of Alibaba Cloud load balancers can lead to remote code execution on their products. We contacted LDPlayer and BigNox again on August 25, 2026, but have not received a response.

Other domains We also disclosed to affected operators outside Alibaba Cloud. Our April 2026 scan identified 640 vulnerable domains, 224 of them hosted by Alibaba Cloud. In August 2026, we re-scanned the remaining 416 and found 100 still vulnerable. To obtain contact addresses, we first looked for a `security.txt`; if none was published, we scraped the site for email addresses and used a large language model to select the most likely candidate. Where both approaches failed, we fell back to `support@domain`. We wrote an email to all 100 affected domains on August 21, 2026. Half of them bounced: 31 were rejected immediately, either because the address did not exist or because the receiving mail server denied our sender, and another 19 failed after repeated delivery attempts over the following days. Of the 50 emails that were delivered,

⁴TLS Security Policies from Alibaba Cloud, archived on August 6, 2026: <https://archive.is/BCRt7>

only three operators replied to us as of August 30, 2026. One acknowledged our report and stated that they would forward it to the responsible team, one closed the corresponding support ticket, and one redirected us to the in-app feedback form of their customer support. None of the three confirmed that they addressed the issue, and we received no reply from the remaining 47 operators.

We will follow a 90-day disclosure window from the date of initial contact, consistent with industry practice. We will release our scanner and proof-of-concept code only after this window has elapsed for the relevant parties. We will not provide any datasets enumerating the vulnerable domains.

Browser Vendors We report the timeline of our disclosure to the three browser vendors below. All dates are in 2026.

- Firefox
 - **April 13:** We report the TLS handshake timeout issue to the Firefox security team.
 - **April 15:** Firefox confirms the report, files a tracking bug in their bug tracker, and adds us to it.
 - **April 16 to April 22:** We answer follow-up questions in the bug tracker.
- Safari
 - **May 8:** We file a report through Apple’s security research portal.
 - **May 11:** Apple responds to our report.
- Chrome
 - **May 21:** We report the issue to Chrome through their public issue tracker.
 - **May 23:** Chrome triages the report and closes it as a duplicate of an existing issue tracking the removal of RSA key exchange. They argue that the root cause is server side and not actionable in the browser.

Internet-wide Measurements. Our scan of the Tranco Top 1M domains is limited to TLS handshakes and to the minimum number of probes required to determine whether a host exposes a padding oracle. We rate-limit our scans, scan from a single source identifiable by reverse DNS, and provide an abuse contact and an opt-out mechanism on that host. For latency measurements via the Globalping network, we issue only ICMP and TCP probes that mirror standard reachability tests. We retain raw scan results in access-controlled storage and publish only aggregated statistics to prevent mapping back to vulnerable domains.

References

- [1] D. Bleichenbacher, “Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1,” in *CRYPTO*, 1998.
- [2] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni *et al.*, “DROWN: Breaking TLS Using SSLv2,” in *USENIX Security Symposium*, 2016.
- [3] H. Böck, J. Somorovsky, and C. Young, “Return Of Bleichenbacher’s Oracle Threat (ROBOT),” in *USENIX Security Symposium*, 2018.
- [4] H. Kario, “Everlasting robot: the marvin attack,” *ESORICS*, 2023.
- [5] J. Manger, “A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS#1 v2.0,” in *CRYPTO*, 2001.

- [6] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J.-K. Tsay, “Efficient Padding Oracle Attacks on Cryptographic Hardware,” in *CRYPTO*, 2012.
- [7] R. Merget, J. Somorovsky, N. Aviram, C. Young, J. Fliegenschmidt, J. Schwenk, and Y. Shavitt, “Scalable scanning and automatic classification of TLS padding oracle vulnerabilities,” in *USENIX Security Symposium*, 2019.
- [8] E. Rescorla and T. Dierks, “The Transport Layer Security (TLS) Protocol Version 1.2,” RFC 5246, August 2008. [Online]. Available: <https://doi.org/10.17487/RFC5246>
- [9] T. Jager, J. Schwenk, and J. Somorovsky, “On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption,” in *ACM CCS*, 2015.
- [10] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero, “Coming of Age: A Longitudinal Study of TLS Deployment,” in *ACM IMC*, 2018.
- [11] D. Warburton, “The 2021 TLS Telemetry Report,” F5 Labs, 2021. [Online]. Available: <https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report>
- [12] E. Ronen, R. Gillham, D. Genkin, A. Shamir, D. Wong, and Y. Yarom, “The 9 Lives of Bleichenbacher’s CAT: New Cache ATtacks on TLS Implementations,” in *IEEE S&P*, 2019.
- [13] L. Capol, “Experimenting with the Bleichenbacher Attack,” Bachelor’s Thesis, ETH Zürich, 2021. [Online]. Available: https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/appliedcrypto/education/theses/bachelors-thesis_livia-capol.pdf
- [14] D. Lenski, “Surveying Public Keys Used on the Internet,” The Cryptography Caffè, SandboxAQ, 2025. [Online]. Available: <https://cryptographycaffe.sandboxaq.com/posts/survey-public-keys-internet/>
- [15] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, “Tranco: A research-oriented top sites ranking hardened against manipulation,” in *NDSS*, 2018.
- [16] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, p. 120–126, February 1978. [Online]. Available: <https://doi.org/10.1145/359340.359342>
- [17] M. Bellare and P. Rogaway, “Optimal asymmetric encryption,” in *Advances in Cryptology — EUROCRYPT ’94*, ser. Lecture Notes in Computer Science, A. D. Santis, Ed. Springer, Berlin, Heidelberg, 1995, vol. 950, pp. 92–111. [Online]. Available: <https://doi.org/10.1007/BFb0053428>
- [18] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, “PKCS #1: RSA Cryptography Specifications Version 2.2,” RFC Editor, Request for Comments 8017, November 2016. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc8017.html>
- [19] V. Klíma, O. Pokorný, and T. Rosa, “Attacking RSA-based sessions in SSL/TLS,” in *CHES*, 2003.
- [20] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, August 2018. [Online]. Available: <https://www.rfc-editor.org/info/rfc8446>
- [21] Nox Limited, “NoxPlayer,” 2026. [Online]. Available: <https://www.bignox.com/>
- [22] LDPlayer, “LDPlayer,” 2026. [Online]. Available: <https://www.ldplayer.net/>

An extended abstract of this paper is in the proceedings of NDSS 2027. This is the full version.

- [23] ZMap Project, “ZGrab2,” 2026. [Online]. Available: <https://github.com/zmap/zgrab2>
- [24] Alban Diquet, “SSLyze,” 2026. [Online]. Available: <https://github.com/nabla-c0d3/sslyze>
- [25] Globalping, “Globalping - Internet and web infrastructure monitoring and benchmarking,” <https://globalping.io>, 2022-2026, accessed: 2026-04-30.

A Generative AI Usage Disclosure

We used Claude to assist in writing code for Section 8, as well as initial assistance for the attack code. All generated code was manually reviewed and experimental results were validated by the authors. We used Claude for the initial draft for Section 4. The generated text was then manually reviewed and evaluated for errors by the authors. We then used ChatGPT for polishing the prose of Section 4 and Appendix H once the full proofs were finalized by the authors. We used Codex for generating the figures in Appendix F. We used Grammarly for grammar and spell checking of the text.

B Open Science

To reproduce the core claims of the paper, our artifact contains:

- The source code of our sequential and parallel implementation.
- A script to simulate Reverse-Bleichenbacher for different oracles to reproduce Table 2, Figure 4, and the additional configurations in Appendix E.
- A script to perform Reverse-Bleichenbacher under different network latencies to reproduce Figure 5.
- The Internet-wide scanning pipeline (ZMap, ZGrab2, and padding-oracle scanners) used in Section 8, including the configuration files and the analysis scripts that produce the reported scan numbers.
- The Tranco Top-1M hosting-provider classification dataset and plotting script used to reproduce Figure 6.
- The Globalping-based latency measurement scripts used to derive the round-trip-time statistics in Section 8.
- The browser timeout and client retry test harness used to reproduce Section 8.4.

Access. The artifact can be accessed through <https://github.com/cispa/ReverseBleichenbacher>

C Mitigations

Countermeasures against the Bleichenbacher attack are well studied, and our work does not change the guidance on implementing PKCS#1 v1.5 unpadding securely. It does, however, change which *deployment-level* mitigations remain effective. We therefore revisit the countermeasures proposed by Ronen et al. [12] in light of Reverse-Bleichenbacher.

Handshake Timeouts. Ronen et al. recommend that clients enforce short TLS handshake timeouts. At an intra-region RTT of 1.0 ms, our controlled measurements give a median sequential attack time of 4.5 s (Figure 5). Excluding even this favorable median-case attack requires timeouts below the duration of a legitimate handshake over a slow link.

Speed Limitation. Ronen et al. propose limiting the rate of RSA decryptions, assuming RSA key exchange is a negligible share of TLS traffic; this does not hold for the shared TLS terminators that expose oracles (Section 7), which serve legacy clients precisely because of RSA. A more targeted variant remains useful: servers can instead limit PKCS#1 v1.5 *decryption failures* per key, which legitimate clients rarely trigger.

Large RSA Keys. Ronen et al. recommend keys of at least 2048 bit, since Bleichenbacher-type attacks require on the order of $\log N$ consecutive oracle calls. As Reverse-Bleichenbacher recovers only the 368 bit PreMasterSecret (Section 4.3), the number of *sequential* queries no longer scales with $\log N$: the 4096 bit load balancer in our case study is exploitable end-to-end.

Best Practices. The remaining countermeasures are unaffected by our results and continue to apply unchanged. Servers that no longer require backward compatibility should disable RSA key exchange entirely, for example by restricting the deployment to TLS 1.3 [20],⁵ and clients should stop negotiating TLS_RSA ciphersuites. Deployments that must retain RSA key exchange should separate certificates and avoid sharing keys across endpoints, TLS versions, and services. Implementations must handle padding failures uniformly by continuing the handshake with a randomly generated PreMasterSecret [8], and must apply the same treatment to the client version check to avoid a Bad Version Oracle [19]. As uniform error handling only covers explicit oracles, the decryption path must additionally be constant-time. Implementations should be validated against timing-oracle test suites such as `tlsfuzzer`, since such leakage has repeatedly been found in mainstream libraries [4].

D Oracle Type of the Alibaba Cloud Load Balancer

Section 2.3 argues that FFT is the oracle type an attacker should expect against an unknown target. The Alibaba Cloud load balancer of Section 7 is one such target, and we determined its type by measurement. On August 12, 2026, we provisioned a fresh Alibaba Cloud load balancer, added an HTTPS listener with its default TLS configuration, and probed it with the ROBOT detection tool [3], slightly modified, so it can deal with inconsistent output. Figure 7 shows the output.

The scanner sends one well-formed `ClientKeyExchange` and four malformed ones, and reports which of the five responses are distinguishable. Bad request 1 replaces the `0x00 || 0x02` prefix, and bad request 3 removes every zero byte after the padding; both produce a response distinct from the well-formed request, so the balancer enforces the prefix and the separator byte. Due to the separator byte requirement, it is an FFT oracle.

E Additional Timeout Feasibility Results

Figures 9 and 10 give the configurations omitted from the main feasibility discussion. At 2048-bit TTT, Capol and our attack finish within the query budget in 39.9% and 86.5% of runs. At 1024 bits, their respective rates are 71.8% and 87.5% for TTT, 68.4% and 86.6% for TFT, and 0.0% and 60.3% for FFT.

F Trimmers Discussion

We now discuss the empirical behavior of Trimmers in more detail, and separate the failure modes of Table 3 by how large a trimmer the plaintext admits.

The main point is that the trimming step introduces an inherent probabilistic failure mode: for any bounded search over small divisors, a random padded message may simply not contain a

⁵<https://configurator.tlsref.org>

An extended abstract of this paper is in the proceedings of NDSS 2027. This is the full version.

```
$ robot-detect <redacted>
Scanning host <redacted> ip <redacted> port 443
RSA N: 0xe107b0baeafc9b6eb0348d63fbef68ba...e11e651
RSA e: 0x10001
Modulus size: 2048 bits, 256 bytes
The oracle is strong, real attack is possible
VULNERABLE! Oracle (strong) found on <redacted>,
  TLSv1.2, standard message flow
Result of good request:
  TLS alert 20 of length 7
Result of bad request 1 (wrong first bytes):
  No data received from server
Result of bad request 2 (wrong 0x00 position):
  TLS alert 20 of length 7
Result of bad request 3 (missing 0x00):
  No data received from server
Result of bad request 4 (bad TLS version):
  TLS alert 20 of length 7
```

Figure 7: Output of the ROBOT detection tool [3] against a load balancer we provisioned on Alibaba Cloud on August 12, 2026. The host and its address are redacted, the modulus is truncated, and long lines are wrapped to fit the column; the reported results are unmodified. Bad requests 1 and 3 are distinguishable from the well-formed request, which identifies the balancer as an FFT oracle.

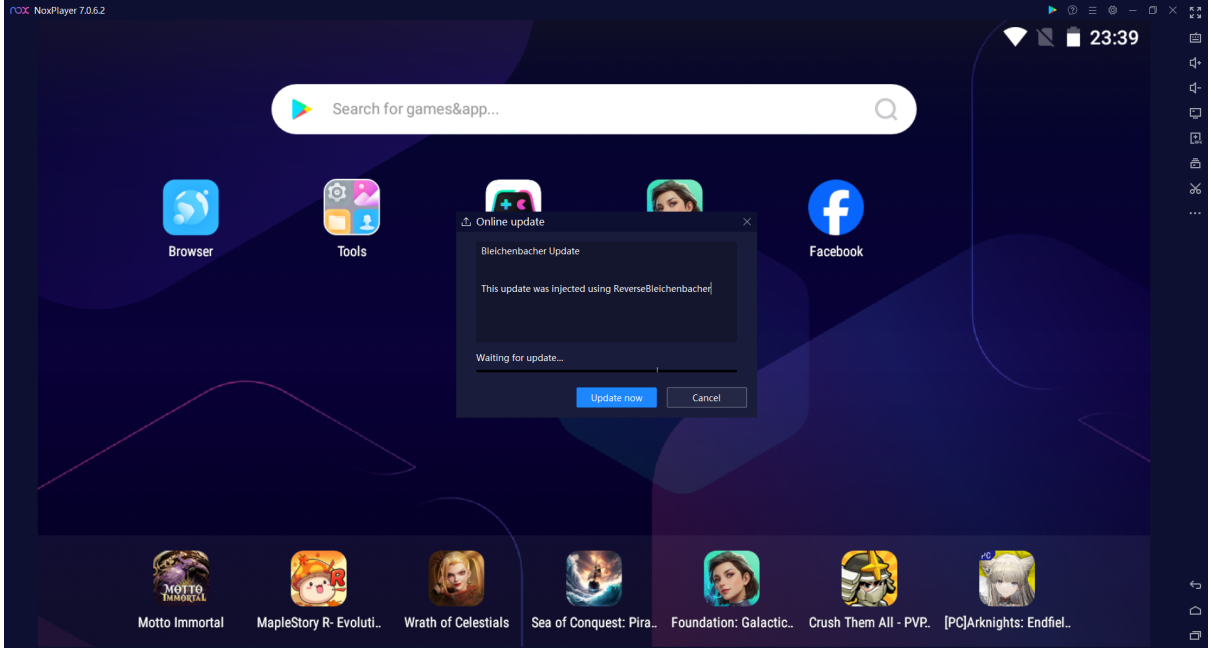


Figure 8: Screenshot of BigNox Android emulator while an attacker injects a malicious update using Reverse-Bleichenbacher. The update description is fully attacker-controlled. When the user clicks **Update Now**, an executable is downloaded from an attacker-chosen URL and then executed. Windows SmartScreen might additionally prompt the user before execution.

useful divisor in the tested range. This limitation is independent of implementation details and oracle noise, and must be kept apart from the failures that oracle noise does cause.

Figure 11 quantifies this effect for the deterministic sample of PKCS#1 v1.5-encoded plaintexts used by our 2048-bit experiments. For each pair (L, T) , the heatmap reports the fraction of messages satisfying

$$\tau_L := \gcd(m, \text{lcm}(1, \dots, L)) \geq T,$$

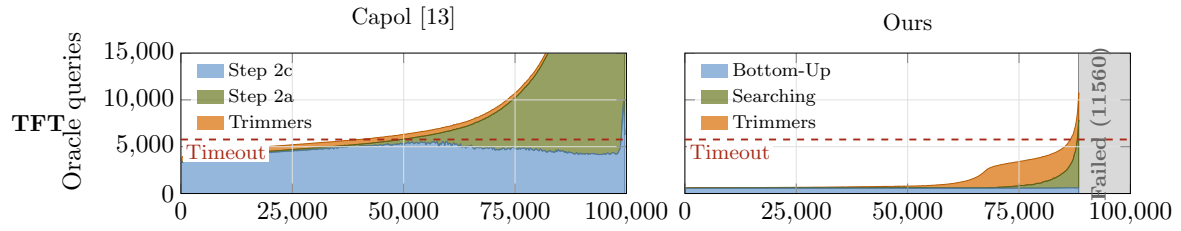


Figure 9: Per-phase query distributions for 2048-bit TFT. Plot format and timeout budget follow Figure 4.

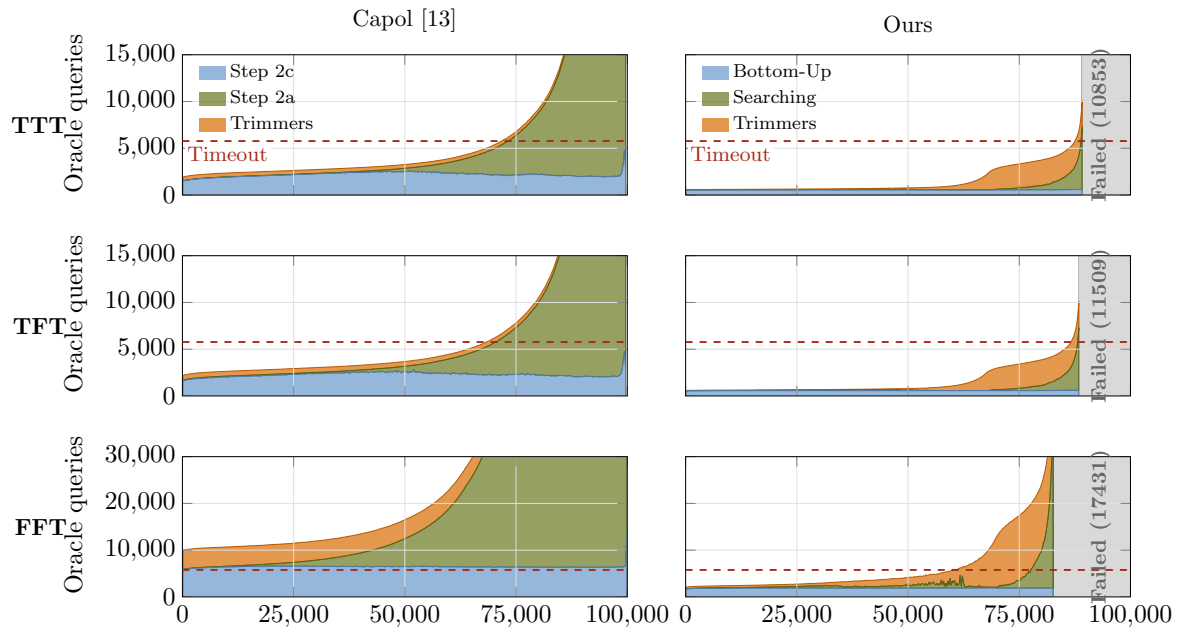


Figure 10: Per-phase query distributions for all 1024-bit oracle types. Plot format and timeout budget follow Figure 4.

Table 3: Attack outcomes over 100 000 runs per configuration (percent). Unresolved includes any unclassified Search or Bottom-Up abort. Rows sum to 100 before rounding; Σ gives subtotals.

Oracle	Key size	Success	Intrinsic (plaintext)			Oracle-induced (false neg.)			Unresolved
			No div.	No mult.	Σ	Missed trim.	Search FN	Σ	
TTT	1024	89.15	8.96	1.88	10.84	0	0	0	0.01
	2048	89.04	9.10	1.84	10.94	0	0	0	0.01
TFT	1024	88.49	8.96	1.88	10.84	0.24	0.41	0.66	0.01
	2048	88.44	9.10	1.84	10.94	0.23	0.38	0.60	0.01
FFT	1024	82.57	8.96	1.88	10.84	4.79	1.79	6.58	0.01
	2048	84.80	9.10	1.84	10.94	3.21	1.04	4.25	0.01

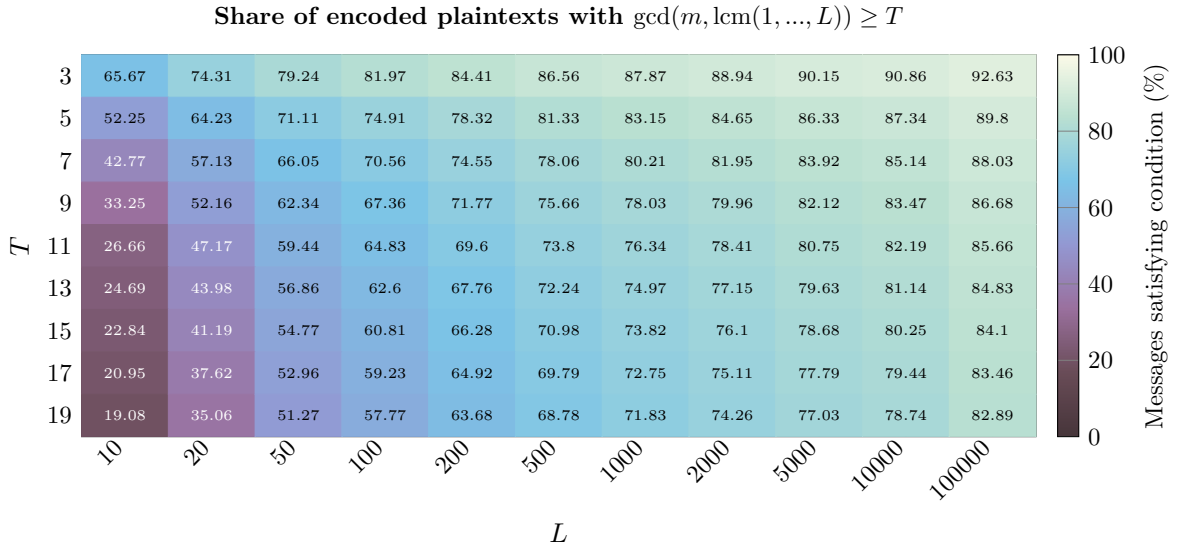


Figure 11: Heatmap of the divisibility upper bound $\gcd(m, \text{lcm}(1, \dots, L)) \geq T$ for PKCS#1 v1.5-encoded plaintexts with 2048-bit RSA keys.

that is, the fraction whose divisibility permits a trimmer of size at least T from candidates up to L . Even at $L = 10\,000$, which would already require a large number of trimmer queries, only 90.86 % of messages satisfy $\tau_L \geq 3$. Increasing the bound to $L = 100\,000$ raises this fraction only to 92.63 %. Thus, 7.37 % to 9.14 % of these encoded plaintexts still lack even this smallest useful divisor under exhaustive small-divisor search, before accounting for candidates for which no tested multiplier preserves conformance. This bound is what makes the intrinsic failure family of Table 3 irreducible in practice: it is set by the plaintext distribution and the candidate range, not by the oracle or the implementation.

Conditioning outcomes on the available trimmer. We study how the attack outcome depends on the trimmer divisors present in the target plaintext. For the candidate divisors $\{t_i\}$ configured for each oracle, we define the *available trimmer* of a plaintext m as

$$\tau_{\text{avail}} = \gcd(m, \text{lcm}(t_1, \dots, t_k)),$$

This is an offline divisibility upper bound on the trimmer the attack can accumulate; a conforming multiplier may not exist for every divisor. The TTT and FFT candidate sets differ, so paired plaintexts need not fall in the same τ_{avail} bin.

Figure 12 shows the success probability and query costs for each oracle type, conditioned on the available trimmer τ_{avail} of a run. The TTT failures are confined to small available trimmers,

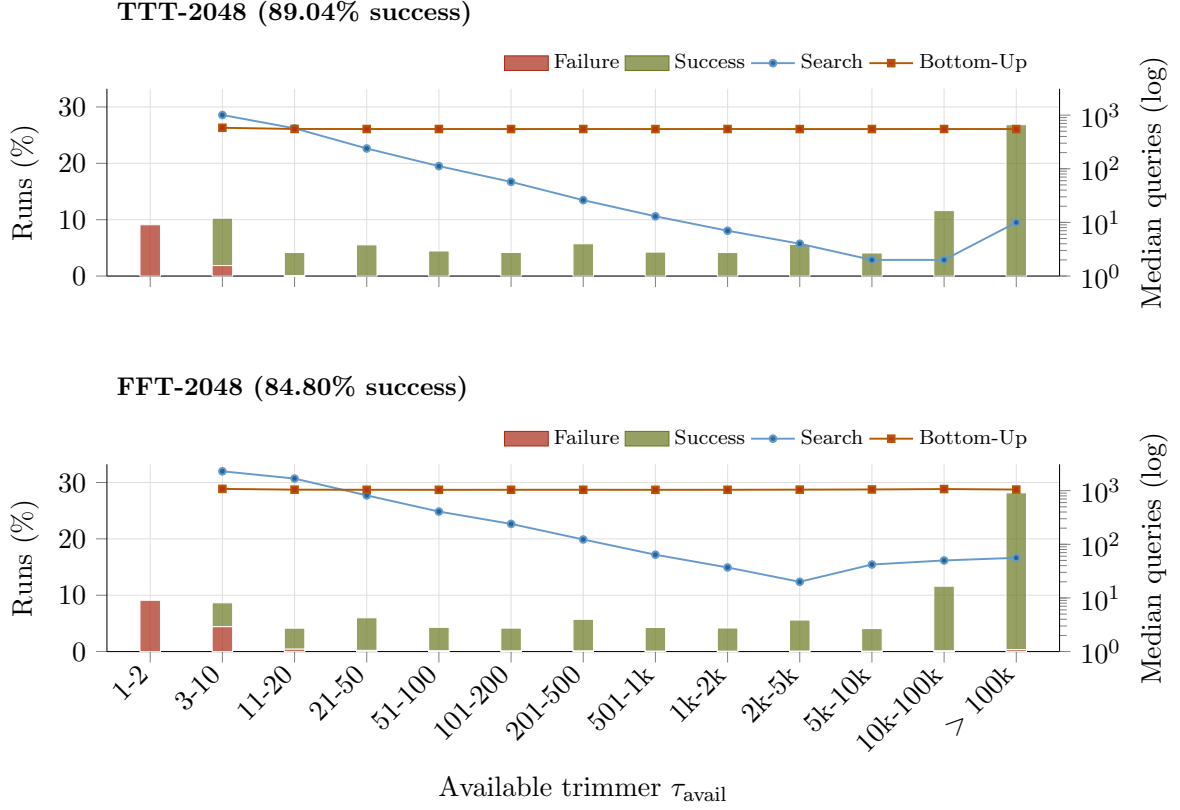


Figure 12: Distribution of τ_{avail} and query costs over 100 000 runs per oracle in our 2048-bit simulations. Stacked bars show the fraction of all runs in each available-trimmer bin, split by attack outcome; lines show the median Searching and Bottom-Up query counts among successful runs in the same bins.

whereas FFT failures remain visible at larger values because oracle false negatives can defeat a run even when the plaintext admits a large trimmer. It is also possible to see that better trimmers positively impact the query cost of the Searching step.

Table 4 gives a finer breakdown of the attack outcome conditioned on τ_{avail} , distinguishing failures attributed to Trimmers from later-phase failures.

Three observations follow, corresponding to the dominant modes of Table 3. First, the bins $\tau_{\text{avail}} \leq 2$ fail with certainty under every oracle, TTT included. These 9.10 % of runs are the *no divisor* mode: no candidate is usable, so there is nothing for any oracle to confirm. Second, the bins $\tau_{\text{avail}} \in \{3, 4\}$ fail even under the noiseless TTT oracle at 74.57 % and 24.58 %, respectively. Most $\tau_{\text{avail}} = 3$ and $\tau_{\text{avail}} = 4$ failures are *no multiplier* cases: the divisor is there, but every tested Trimmers offset takes the message outside $[2B, 3B)$. Together with the no-divisor cases, these account for nearly all TTT failures in this sample. Rare Bottom-Up aborts remain unresolved, including at $\tau_{\text{avail}} \geq 5$. Third, almost all FFT failures in excess of the corresponding TTT failures are oracle-induced; rare Bottom-Up aborts remain unresolved. The total FFT failure rate falls with τ_{avail} , from 88.17 % at $\tau_{\text{avail}} = 3$ to 1.20 % above 1000. The later-step failure rate is highest at small τ_{avail} ; above $\tau_{\text{avail}} = 20$ it is at most 0.70 % of runs in each bin.

This also shows how conservative the sufficient condition in the analysis is. For the TTT oracle, nearly every simulated run with $\tau_{\text{avail}} \geq 5$ succeeds, even though the proof uses the sufficient threshold $\tau \geq 13$ for the achieved trimmer, together with the payload-length, modulus-size, and initial-modulus conditions in Section 4.4. The available trimmer is an offline upper bound; the achieved trimmer can be smaller, and the optimized Bottom-Up strategy is best-effort.

Table 4: Outcomes of the attack at 2048 bits conditioned on τ_{avail} , the divisibility upper bound for each oracle’s configured candidate set. We report the fraction of all 100 000 runs in each available-trimmer bin, split by attack outcome and failure mode. Failures are grouped by attributed cause: Trimmers (including a missed factor of two detected in Searching) or a later step (Searching or Bottom-Up). Outcome percentages are relative to the bin.

τ_{avail}	TTT				FFT			
	All runs (%)	Suc-cess	Fail trim.	Fail later	All runs (%)	Suc-cess	Fail trim.	Fail later
1	6.08	0.00	100.00	0.00	6.08	0.00	100.00	0.00
2	3.02	0.00	100.00	0.00	3.02	0.00	100.00	0.00
3	1.99	25.43	74.32	0.25	1.99	11.83	82.98	5.19
4	1.53	75.42	24.05	0.52	1.53	36.77	52.15	11.08
5	1.46	100.00	0.00	0.00	1.17	56.90	33.25	9.85
6–10	5.28	100.00	0.00	0.00	3.97	69.98	24.35	5.67
11–20	4.17	99.98	0.00	0.02	4.15	89.74	9.63	0.63
21–100	9.98	100.00	0.00	0.00	10.30	96.59	3.22	0.18
101–1000	14.16	100.00	0.00	0.00	14.17	98.12	1.74	0.13
>1000	52.33	100.00	0.00	0.00	53.62	98.80	0.51	0.70

For the FFT oracle, the success rate instead increases with τ_{avail} : small available trimmers remain fragile under false negatives, while the bins from $\tau_{\text{avail}} \geq 21$ upwards stabilize around 97 % to 99 % success. This matches the role of the trimmer in the algorithm: a larger τ_{avail} gives Trimmers the opportunity to return a larger τ , which shrinks the initial interval, improves the granularity of the coverage-driven search, and makes it easier for Bottom-Up to start from a sufficiently narrow interval. It does not eliminate failures when the oracle itself remains noisy.

Impact of the available trimmer on query cost. Table 5 focuses on the cost of these small-available-trimmer cases among the runs that do succeed. For TTT, the median Searching cost tends to decrease as τ_{avail} grows from 3 to 10, and the Bottom-Up cost drops sharply once $\tau_{\text{avail}} \geq 6$. For FFT, the Searching distribution is much more variable, as shown by the large standard deviations, but Bottom-Up still generally decreases with larger τ_{avail} . The FFT means exceed their medians throughout, which is the same effect the failure classification isolates: a run that keeps losing queries to false negatives either recovers late and lands in the upper tail, or exhausts the step and leaves the table as an oracle-induced failure.

G All Searching Optimizations

We discuss in more detail here the two optimizations we apply to the Searching step, which are crucial for reducing the number of queries needed to find an accepting multiplier.

The main goal is still to find some multiplier s such that $m' \cdot s$ hits the target oracle range, which gives us a narrower interval. The difficulty is that large multipliers make such positive answers rare. Multiplying the current interval by s maps it to a modular arc of width approximately $s(b - a)$: as s grows, the interval is stretched more, the hit probability decreases, and each query becomes less useful.

Our two optimizations address exactly this problem and reduce the number of queries needed to find an accepting multiplier. Unlike the simpler standard-oracle search analyzed in Section 4.2, the optimized variant below first changes the effective target range and then uses coverage-driven stepping.

Minimizing stretching. We first replace the original target oracle range by an equivalent one that lies closer to the current interval, which reduces the required multiplier s and therefore the amount of stretching.

Table 5: Query counts for successful 2048-bit runs with small available trimmers. Medians, means, and standard deviations are computed among successful runs with each exact value of τ_{avail} . The corresponding failure-inclusive demographics and outcome rates are reported in Table 4.

Oracle	τ_{avail}	Search median	Search mean $\pm$ sd	Bottom-Up median	Bottom-Up mean $\pm$ sd
TTT	3	1307	1539.6 $\pm$ 1123.8	708	792.0 $\pm$ 270.8
	4	1620	1911.0 $\pm$ 1400.0	626	679.8 $\pm$ 229.0
	5	1205	1741.0 $\pm$ 1657.7	565	571.7 $\pm$ 41.7
	6	1531	1921.5 $\pm$ 1678.6	585	588.0 $\pm$ 26.9
	7	1054	1323.0 $\pm$ 1289.2	604	601.8 $\pm$ 28.6
	8	618	893.8 $\pm$ 933.6	565	580.0 $\pm$ 35.2
	9	598	926.3 $\pm$ 1043.7	563	575.1 $\pm$ 32.4
	10	733	895.4 $\pm$ 879.0	557	563.7 $\pm$ 23.5
FFT	3	1519	1684.4 $\pm$ 1177.7	1208	1299.3 $\pm$ 326.0
	4	2119	2503.1 $\pm$ 1878.7	1119	1165.6 $\pm$ 192.1
	5	2616	3384.8 $\pm$ 2789.6	1066	1071.9 $\pm$ 66.8
	6	2511	3751.8 $\pm$ 3399.5	1071	1071.7 $\pm$ 47.2
	7	2314	3605.6 $\pm$ 3695.9	1082	1084.3 $\pm$ 46.6
	8	2567	3941.1 $\pm$ 3933.3	1082	1082.5 $\pm$ 46.0
	9	2343	3854.3 $\pm$ 4458.8	1070	1076.2 $\pm$ 72.5
	10	2115	3786.0 $\pm$ 4621.8	1057	1058.8 $\pm$ 45.5

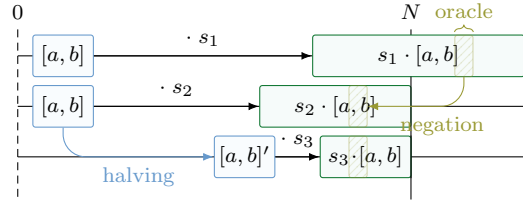


Figure 13: Minimizing stretching. Each row shows the interval $[a, b]$ multiplied by the corresponding multiplier into the target range (striped). In the standard attack (top), the large distance requires a large s_1 , heavily stretching the interval. Negation shifts the oracle closer to $[a, b]$, reducing s_2 . Halving then moves $[a, b]$ near $N/2$, making s_3 minimal.

Coverage-driven stepping. We then use the divisor τ to step in increments of $1/\tau$ in the original oracle, which corresponds to integer steps for the reduced message m' , and partition the current interval into slices so that each query covers a previously-uncovered slice.

G.1 Minimizing Stretching

The first optimization reduces the distance between the current interval and the target oracle range. Since the first useful multiplier is roughly proportional to this distance, moving the interval closer directly reduces the required stretching and increases the hit probability.

For example, in our working case of PKCS#1 v1.5 padding, we have $N \approx 2^{16}B$, so the initial multiplier is on the order of $s \approx 2^{14}$: each query stretches the interval by a factor of approximately 16,000 and yields a hit probability of roughly $1/16,000$.

We now describe the two transformations that reduce this distance, also shown in Figure 13. **Negation.** Notice that with a standard query $\mathcal{O}_{m'}(s)$, we test if $s \cdot m' \in [N + 2B, N + 3B)$ as integers, for the case of small s , which is what we are interested in. However, querying $\mathcal{O}_{m'}(-s)$ instead gives the test

$$N - 3B < m' \cdot s \leq N - 2B.$$

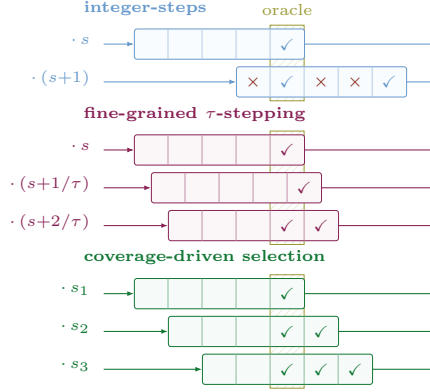


Figure 14: Coverage-driven stepping. Each row maps $s_j \cdot [a, b]$ into the transformed target range (striped); a check marks a tested slice. Integer-stepping misses slices, fine-grained τ -stepping overlaps, and coverage-driven selection tiles $[a, b]$ exactly.

This effectively moves the oracle range closer to the working interval, lowering the multipliers we seek from around $\frac{N+2B}{b}$ to around $\frac{N-3B}{b}$.

Halving. The trimming step guarantees that $m' = m/\tau$ is odd, since all factors of 2 have been absorbed into τ . Multiplication by $2^{-1} \bmod N$ separates even and odd residues into the lower and upper halves of the modular line (recall Figure 1). Since m' is odd, we know that $m' \cdot 2^{-1} \bmod N$ lies in the upper half $[N/2, N)$, which halves the remaining distance between our working interval and the oracle range at no query cost. More precisely, let t be an integer and write $s = t + 1/2$ for the effective real multiplier. The actual oracle query uses $-(t + \hat{2}) \bmod N$, where $\hat{2} = 2^{-1} \bmod N$. Since m' is odd,

$$m'(t + \hat{2}) \equiv \frac{N}{2} + sm' \pmod{N}.$$

Thus on the searched branch, a successful query means $m' \cdot s \in (N/2 - 3B, N/2 - 2B]$. Effective multipliers lie in $\mathbb{Z} + 1/2$, with consecutive values spaced by 1.

After negation and halving, the relevant distance drops from about N to about $N/2 - 5B$. Consequently, the first accepting multiplier is on the order of $N/(2b)$ rather than N/b , and one positive query already yields a sub-interval of width about $b \cdot 2B/N$.

G.2 Coverage-Driven Stepping

We now scan this reduced search space to find an accepting effective multiplier. Viewed in terms of the reduced message m' , this is simply the brute-force search of the original Step 2a, but carried out on a smaller interval.

We describe below and in Figure 14 how knowledge of τ lets us perform this search more finely and efficiently, i.e., how to find an effective multiplier s such that $m' \cdot s \in (N/2 - 3B, N/2 - 2B]$.

Why integer steps fail. Step 2a’s classical search increments the multiplier by 1 at each query (Figure 14, top). In the original setting, this starts from the hypothesis that $m \in [2B, 3B)$, so increasing the multiplier by 1 shifts the queried interval by at least $2B$. This distance is wider than the oracle range B , which misses slices.

Fine-grained τ -stepping. The trimming step divided the original message by τ , producing a reduced message $m' = m/\tau$ in a smaller interval $[a, b]$, where $b \leq 3B/\tau$. This means that integer steps with respect to the reduced message m' shift the queried interval by around $3B/\tau$ at a time, and for sufficiently large τ this is less than the width of the transformed target range. Thus, querying the oracle with these values $s, s + \tau^{-1}, s + 2 \cdot \tau^{-1}, \dots$ restores the granularity needed to land inside that range. However, the same fineness means that consecutive multipliers cover almost the same slice of $[a, b]$, so testing every one of them is wasteful.

Coverage-driven selection. The fix is to test only those multipliers whose preimages tile $[a, b]$ without overlap (Figure 14, bottom). For integer endpoints, we partition $[a, b]$ into consecutive integer slices $[a_1, b_1], \dots, [a_J, b_J]$ with $b_1 = b$ and $b_{j+1} = a_j - 1$. For each slice, we pick a multiplier s_j such that

$$s_j \cdot x \in (N/2 - 3B, N/2 - 2B]$$

for every $x \in [a_j, b_j]$. We build the slices greedily from top to bottom: given b_j , the multiplier

$$s_j = \left\lfloor \frac{N/2 - 2B}{b_j} - \frac{1}{2} \right\rfloor + \frac{1}{2}$$

is the largest admissible half-integer that maps b_j below the top of the transformed target range. The slice extends down to $a_j = \max(a, \lfloor (N/2 - 3B)/s_j \rfloor + 1)$. For $b \leq B$, the top point b_j is accepted, so this gives a nonempty slice. A positive answer on s_j then identifies $[a_j, b_j]$ as the new interval for m' .

Query count. After the distance reduction of Appendix G.1, relevant multipliers near a point x satisfy $s_j \approx N/(2x)$, so the slice covered by s_j has width about $B/s_j \approx x \cdot 2B/N$. The slice widths therefore scale linearly with position: the first slice has width about $b \cdot 2B/N$, and after J slices the lower endpoint is

$$a_J \approx b \left(1 - \frac{2B}{N}\right)^J \approx b \cdot e^{-2BJ/N}.$$

Reaching a thus requires

$$Q_{\text{search}} \approx \frac{N}{2B} \log \frac{b}{a} \quad (7)$$

queries. The resulting accepting slice has width at most B/s_j . Accounting for the spacing of admissible multipliers gives the exact bound

$$b_j - a_j \leq \frac{B}{s_j} \leq \frac{b_j B}{N/2 - 2B - b_j} \leq \frac{bB}{N/2 - 2B - b},$$

provided $b < N/2 - 2B$. Thus $b \cdot 2B/N$ is an approximation, not an exact upper bound.

This step approximately halves the coverage query count from Section 4.2 at the cost of approximately doubling the width of the interval returned to Bottom-Up. We next make this tradeoff precise and verify that the wider interval still satisfies the Bottom-Up conditions.

G.3 Integration with the Full Algorithm

Use the notation from Section 4.4: after Trimmers, the reduced message lies in

$$[a', b'] = \left[\left\lceil \frac{2B}{u_l} \right\rceil, \left\lfloor \frac{3B - 1}{u_h} \right\rfloor \right].$$

Halving and coverage-driven stepping return an accepting slice $[\tilde{a}, \tilde{b}] \subseteq [a', b']$. The exact rounded bound above gives

$$\tilde{b} - \tilde{a} \leq \frac{b'B}{N/2 - 2B - b'} \leq \frac{6B^2}{(N - 10B)u_h},$$

where we used $b' \leq 3B/u_h \leq 3B$. Under the sufficient trimming condition $\tau \geq 13$, we also have $b' < B$, so every coverage slice is nonempty.

Since $\tilde{b} \geq a' \geq 2B/u_l$, it follows that

$$\frac{\tilde{b}}{\tilde{b} - \tilde{a}} \geq \frac{N - 10B}{B} \cdot \frac{u_h}{3u_l} \geq \frac{N - 10B}{3B} \geq \frac{N}{4B}.$$

The last inequality uses $u_h \geq u_l$ and $N \geq 256B > 40B$. Thus, despite the larger final interval, the halving search satisfies the width condition in Theorem 3 for the initial modulus $D_0 = 16$.

For the sequential schedule $d_0 = 4$ and $\ell = 1$, the remaining hypotheses are the same as in the standard full algorithm. There are at most eight candidates in a Bottom-Up round, so $8 \leq N/B$ suffices. Moreover, $\tilde{b} \leq b' \leq 3B/u_h$, and therefore

$$2^n + 2\tilde{b} \leq 2^n + \frac{6B}{u_h} \leq \frac{B}{2}$$

whenever $2^n \leq B/26$ and $u_h \geq \tau \geq 13$. Consequently, with a perfect oracle, a correctly recovered divisor τ for which m/τ is odd, and offsets chosen by the proved construction, halving followed by sequential Bottom-Up recovers the requested suffix under the sufficient conditions

$$\tau \geq 13, \quad 8 \leq N/B, \quad 2^n \leq B/26.$$

The expected number of halving-search queries is

$$Q_{\text{search}} \approx \frac{N}{2B} \log \frac{b'}{a'} \approx \frac{N}{2B} \log \frac{3u_l}{2u_h},$$

approximately half the standard-search estimate from Section 4.4. This estimate describes the complete halving grid alone. The optimized TTT schedule spends sparse-grid queries first and can incur additional queries before reaching that grid when its retries fail. Sequential Bottom-Up still has the worst-case bound $Q_{\text{BU}} \leq 8 + 2(n - 4) = 2n$. Thus halving changes the expected Searching cost but not the proved sequential Bottom-Up bound.

The bounds above are deliberately worst-case. In the simulations from Section 6, nearly all TTT runs with $\tau \geq 3$ succeed, and from $\tau \geq 6$ the Bottom-Up query count is already in line with the empirical estimate $Q_{\text{BU}} \approx 1.5n = 552$ for the TLS PreMasterSecret; see also Figure 12 and Appendix F.

For a batched continuation with $\ell > 1$, the same width argument applies. The other two conditions in Theorem 3 become

$$D_{\max} + 2\tilde{b} \leq B/2, \quad \max(8, 2^\ell) \leq N/B,$$

and Bottom-Up uses at most $8 + 2^\ell \lceil (n - 4)/\ell \rceil$ queries in $1 + \lceil (n - 4)/\ell \rceil$ adaptive rounds.

H Bottom-Up Proofs

We provide the correctness proof for the multiplier construction sketched in Section 4.3. The next subsection proves the existence of some valid parameters r and δ under the interval bounds used in Theorem 2; it does not analyze the specific heuristic choices from Section 4.3.1. In practice, we use the centered heuristic from Section 4.3.1; empirically, it succeeds for all practical parameters we tested. As in Section 4.3, let $x = m'$ denote the reduced working message after trimming.

H.1 Correctness of the Multiplier Construction

Lemma 4. *Suppose that $0 < a \leq b$, $x \in [a, b]$ is an integer, and $D \geq 2$ is an integer with $\gcd(D, N) = \gcd(x, D) = 1$. Suppose that there exist a non-negative integer δ and an integer $r \in [1, D - 1]$ such that*

$$\left[\delta a + \frac{rN}{D}, (\delta + 1)b + \frac{rN}{D} + D \right] \subset (N - 3B, N - 2B].$$

Then, for any $v \in \mathbb{Z}_D^*$, we can construct a single oracle query to $\mathcal{O}_x^-$, using a multiplier $s(v) = \delta + k \cdot \hat{D}$, with the following guarantees.

If we let $v^* = x \bmod D$ denote the correct residue class of x , we have that $\mathcal{O}_x^-(s(v^*)) = 1$, and that $\mathcal{O}_x^-(s(v)) = 0$ if $\|k(v - v^*)\hat{D}\|_N \geq B$, where $k = k(v)$.

Proof. Since we suppose that $x \equiv v^* \pmod{D}$, we can write $x = D \cdot l + v^*$ for some l . Let $\hat{D}$ be the inverse of D modulo N , in particular, let $D \cdot \hat{D} = 1 + hN$ for some positive integer h ; notice that $\hat{D} = \lceil \frac{hN}{D} \rceil$, and $h = \lfloor \hat{D} \cdot D/N \rfloor$.

Now, let δ, r be any integers that satisfy the hypothesis, and in particular

$$\begin{aligned} \delta a + \frac{rN}{D} &> N - 3B \\ (\delta + 1)b + \frac{rN}{D} + D &\leq N - 2B \end{aligned}$$

Since $hN \equiv -1 \pmod{D}$, we have $\gcd(h, D) = 1$. Given a guess $v \in \mathbb{Z}_D^*$, the inverse of vh therefore exists; choose $k = r(vh)^{-1} \bmod D$ in $[0, D)$, so that $k \cdot v \cdot h \equiv r \pmod{D}$.

We claim that

$$kv\hat{D} \bmod N \approx \left\lceil \frac{rN}{D} \right\rceil.$$

Indeed $kv\hat{D}D = kv(1 + hN) = kv + kvhN = kv + (r + \eta D)N$ for some integer η , so $kv\hat{D} = \frac{kv + rN}{D} + \eta N$; since both k and v are less than D , we have that $kv\hat{D} \bmod N$ is distant from $\lceil \frac{rN}{D} \rceil$ by at most D .

We finally construct the multiplier $s(v)$ using the above computed k and the offset δ from hypothesis,

$$s(v) = \delta + k \cdot \hat{D}.$$

Then it holds that

$$\begin{aligned} x \cdot s(v) &= (Dl + v^*)(\delta + k\hat{D}) \\ &= x\delta + klD\hat{D} + kv^*\hat{D} \\ &= x\delta + klD\hat{D} + kv\hat{D} + k(v^* - v)\hat{D} \\ &\equiv x\delta + kl + \frac{kv + rN}{D} + k(v^* - v)\hat{D} \pmod{N}. \end{aligned}$$

Thus, for $v = v^*$, we have

$$x \cdot s \bmod N \in \left[\delta a + \frac{rN}{D}, (\delta + 1)b + \frac{rN}{D} + D \right],$$

and by our choice of r and δ , this indeed implies that $x \cdot s \bmod N \in (N - 3B, N - 2B]$, i.e. that $\mathcal{O}_x^-(s(v)) = 1$.

For an arbitrary guess v , define the auxiliary integer

$$A_v = x\delta + kl + \frac{kv + rN}{D}.$$

It is an integer because $hN \equiv -1 \pmod{D}$ and $kvh \equiv r \pmod{D}$ imply $kv + rN \equiv 0 \pmod{D}$. Since $0 \leq kl \leq x \leq b$ and $0 \leq kv/D < D$, the same interval bounds give $A_v \in (N - 3B, N - 2B]$, even for an incorrect guess. The expansion above gives

$$xs(v) \equiv A_v + k(v^* - v)\hat{D} \pmod{N}.$$

If the query for an incorrect guess were accepted, then A_v and $xs(v) \bmod N$ would both lie in the half-open interval $(N - 3B, N - 2B]$ of length B . Their circular distance would therefore be strictly less than B :

$$\|xs(v) - A_v\|_N < B.$$

The displayed congruence instead gives

$$\|xs(v) - A_v\|_N = \|k(v - v^*)\hat{D}\|_N.$$

Therefore $\|k(v - v^*)\hat{D}\|_N \geq B$ suffices for rejection. $\square$

H.2 Proof of the existence of offsets

Lemma 5. *Suppose that $0 < a < b$ and $D \geq 2$ is an integer. Suppose that*

$$\frac{2B + D + 2b}{D - 1} \leq \frac{N}{D} \leq \frac{b}{b - a} \cdot (B - D - 2b).$$

Then there exist a non-negative integer δ and an integer $r \in [1, D - 1]$ such that

$$\left[\delta a + \frac{rN}{D}, (\delta + 1)b + \frac{rN}{D} + D \right] \subset (N - 3B, N - 2B).$$

If the stronger upper bound

$$\frac{2N}{D} \leq \frac{b}{b - a}(B - D - 2b)$$

holds, then r can be chosen odd.

Proof. We want to find integers $\delta \geq 0$ and $r \in [1, D - 1]$ such that

$$\delta \cdot a + \frac{rN}{D} > N - 3B, \quad \text{and} \quad (\delta + 1) \cdot b + \frac{rN}{D} + D \leq N - 2B.$$

This is equivalent to the following system of inequalities:

$$\begin{aligned} \delta &> \frac{N - 3B - \frac{rN}{D}}{a} \\ \delta &\leq \frac{N - 2B - D - \frac{rN}{D}}{b} - 1 \end{aligned}$$

It is enough to prove the stronger statement that this system has a positive integer solution. If we denote by $R = \frac{rN}{D}$, a sufficient condition is that the upper bound is at least 1 and exceeds the lower bound by at least 1, i.e.

$$\frac{N - 2B - D - R}{b} - 1 \geq \max \left(\frac{N - 3B - R}{a} + 1, 1 \right).$$

The first one is equivalent to, after multiplying by ab ,

$$Na - 2Ba - Da - Ra - ab \geq Nb - 3Bb - Rb + ab,$$

and the second one is $N - 2B - D - R \geq 2b$, which gives us the following interval

$$N - B \cdot \frac{3b - 2a}{b - a} + \frac{Da + 2ab}{b - a} \leq R \leq N - 2B - D - 2b.$$

The existence of such $R = \frac{rN}{D}$ with $r \in [1, D - 1]$ is implied by the following three conditions:

$$\begin{aligned} N - 2B - D - 2b &\geq \frac{N}{D} \\ N - B \cdot \frac{3b - 2a}{b - a} + \frac{Da + 2ab}{b - a} &\leq \frac{(D - 1)N}{D} \\ N - 2B - D - 2b &\geq N - B \cdot \frac{3b - 2a}{b - a} \\ &\quad + \frac{Da + 2ab}{b - a} + \frac{N}{D} \end{aligned}$$

An extended abstract of this paper is in the proceedings of NDSS 2027. This is the full version.

The first one is equivalent to $2B + D + 2b \leq \frac{(D-1)N}{D}$, the second one is equivalent to $B \cdot \frac{3b-2a}{b-a} - \frac{Da+2ab}{b-a} \geq \frac{N}{D}$, and the third one is equivalent to $B \cdot \frac{3b-2a}{b-a} - \frac{Da+2ab}{b-a} \geq 2B + D + 2b + \frac{N}{D}$, which implies the second one.

The first one is given by the hypothesis. Multiplying the last inequality by $D(b-a)$ we get

$$\begin{aligned} BD \cdot (3b-2a) - D^2a - 2abD \\ \geq 2BD(b-a) + D^2(b-a) + 2bD(b-a) + N(b-a) \end{aligned}$$

which is also implied by the hypothesis

$$N(b-a) \leq Db(B-D-2b).$$

Notice that we have explicit bounds

$$\begin{aligned} r_{\min} &= D - \frac{BD}{N} \cdot \frac{3b-2a}{b-a} + \frac{D^2a + 2Dab}{N(b-a)} \\ r_{\max} &= D - \frac{2BD + D^2 + 2Db}{N} \end{aligned}$$

and this argument proves that $[r_{\min}, r_{\max}] \cap [1, D-1] \cap \mathbb{Z} \neq \emptyset$, and for any r in this intersection we can find an appropriate value of δ .

For the odd- r assertion, the stronger upper bound gives

$$r_{\max} - r_{\min} = \frac{Db(B-D-2b)}{N(b-a)} \geq 2.$$

The first endpoint condition above gives $r_{\max} \geq 1$, while the explicit formula gives $r_{\max} < D$. Let r be the largest odd integer with $r \leq r_{\max}$. Then $r \geq 1$, and consecutive odd integers are two apart, so

$$r > r_{\max} - 2 \geq r_{\min}.$$

Since $r < D$ is an integer, we also have $r \leq D-1$. Thus the same construction of δ applies with odd r . $\square$

H.3 Proof of Theorem 3

Proof. We maintain a known residue $X \equiv x \pmod{D_{\text{prev}}}$ with $0 \leq X < D_{\text{prev}}$. Initially $D_{\text{prev}} = 2$ and $X = 1$. In the first round we set $D = D_0$; in each of the subsequent ρ rounds we set $D = 2^\ell D_{\text{prev}}$. In either case the candidates are

$$v = X + zD_{\text{prev}}, \quad z = 0, \dots, g-1, \quad g = \frac{D}{D_{\text{prev}}}.$$

Thus $g = D_0/2$ in the first round and $g = 2^\ell$ afterward. Every modulus used by the procedure satisfies $D_0 \leq D \leq D_{\text{max}}$, and every candidate is odd and hence belongs to $\mathbb{Z}_D^*$.

We first verify that the odd offset required by Lemma 5 exists in every round. The size hypothesis gives $D + 2b \leq B/2$. Therefore

$$D(2B + D + 2b) \leq \frac{5}{2}BD \leq 5B(D-1) \leq N(D-1),$$

where the middle inequality uses $D \geq 2$ and the last uses $N \geq 256B$ in the PKCS#1 v1.5 setting. This is equivalent to

$$\frac{2B + D + 2b}{D-1} \leq \frac{N}{D}.$$

For the other side, the width hypothesis gives

$$\begin{aligned} \frac{b}{b-a}(B-D-2b) &\geq \frac{4N}{D_0B} \cdot \frac{B}{2} \\ &= \frac{2N}{D_0} \geq \frac{2N}{D}. \end{aligned}$$

This is the strengthened upper bound in Lemma 5, so an admissible odd r and a non-negative δ exist. Consequently, Theorem 2 accepts the correct residue candidate v^* .

It remains to reject every candidate $v \neq v^*$. Since D is even and N is odd, $D\hat{D} = 1 + hN$ implies that h is odd. The construction has $khv \equiv r \pmod{D}$; because r, h, v are odd, k is odd as well, and hence kh is a unit modulo D . Let $j_v \in [0, D)$ satisfy

$$j_v \equiv kh(v^* - v) \pmod{D}.$$

The difference $v^* - v$ is a nonzero multiple of D_{prev} modulo D . Multiplication by the unit kh preserves this property, and therefore

$$\min(j_v, D - j_v) \geq D_{\text{prev}}.$$

As in the proof of Lemma 4, define

$$Y_v = \left(\delta + \frac{k}{D}\right)x + \frac{rN}{D}.$$

The offset inequalities place Y_v in the negated oracle interval, and the multiplier construction gives, for some integer t ,

$$xs(v) = Y_v + \frac{j_v N}{D} + tN.$$

Consequently,

$$\|xs(v) - Y_v\|_N = \frac{N}{D} \min(j_v, D - j_v) \geq \frac{ND_{\text{prev}}}{D} = \frac{N}{g} \geq B,$$

where the final inequality follows from $g \leq \max(D_0/2, 2^\ell) \leq N/B$. Two points in the oracle interval of width B have circular distance strictly less than B , so the query for an incorrect candidate cannot be accepted. Exactly the correct candidate passes, and it becomes the known residue for the next round.

The first round uses at most $D_0/2$ queries and establishes $x \bmod D_0$. After the following ρ rounds we know $x \bmod D_{\text{max}}$. Since $D_{\text{max}} \geq 2^n$, this determines $x \bmod 2^n$. The later rounds use at most 2^ℓ queries each, giving at most $D_0/2 + \rho 2^\ell$ queries in $1 + \rho$ adaptive rounds. $\square$

I Notation

Table 6:

Symbol	Meaning
a, b	current interval bounds for the working message
a_j, b_j	endpoints of slice j in coverage-driven search
a', b'	interval bounds after Trimmers and before Searching in the full-algorithm summary
A_v	integer auxiliary accepting point in the generic rejection proof
$\tilde{a}, \tilde{b}$	interval bounds returned by Searching and passed to Bottom-Up in the full-algorithm summary
B	PKCS#1 v1.5 padding constant $B = 2^{8(\kappa-2)}$ for a κ -byte RSA modulus
c	RSA ciphertext
d	RSA private exponent
d_0	number of bits established by the initial Bottom-Up round, with $D_0 = 2^{d_0}$
δ	additive offset in the Bottom-Up multiplier construction
$\delta_{\min}, \delta_{\max}$	lower and upper bounds for admissible values of δ
D	divisor / residue-class modulus in Bottom-Up
D_0	residue-class modulus used in the initial Bottom-Up round
$D_{\max}$	largest Bottom-Up modulus, $D_{\max} = 2^{d_0 + \ell\rho}$
D_{prev}	modulus of the residue already known before a Bottom-Up round
$\hat{D}$	inverse of D modulo N
e	RSA public exponent
ε	small probing offset
g	number of residue candidates in a Bottom-Up round, $g = D/D_{\text{prev}}$
h	quotient in the relation $D \cdot \hat{D} = 1 + h \cdot N$
η	local auxiliary quotient, defined by $kvh = r + \eta \cdot D$
i	Bottom-Up round index
j	slice index in coverage-driven search
j_v	canonical representative in $[0, D)$ of $kh(v^* - v) \bmod D$ in the batched rejection proof
J	number of slices in coverage-driven search
κ	RSA modulus length in bytes
k	guess-dependent factor in $s(v) = \delta + k \cdot \hat{D}$; permutes cluster indices when coprime to D
l	quotient in the decomposition $x = D \cdot l + v^*$
ℓ	number of bits recovered per Bottom-Up round
L, U	lower and upper endpoints of the target oracle interval in Searching, with width $U - L = B$
L_{net}	network latency
M	intended payload before PKCS#1 v1.5 padding

Continued on next page

Table 6: (Continued)

m	padded RSA plaintext
m'	reduced working message after trimming, $m' = m/\tau$
n	number of target bits to recover
N	RSA modulus $N = p \cdot q$
$\ t\ _N$	distance from t to the nearest multiple of N ; the circular distance between u and v is $\ u - v\ _N$
(N, e)	RSA public key
$\mathcal{O}_x(s)$	generic padding oracle on working message x ; concrete instances include $\mathcal{O}_m$ and $\mathcal{O}_{m'}$
$\mathcal{O}_x^-(s)$	negated oracle $\mathcal{O}_x^-(s) = \mathcal{O}_x(-s)$
p, q	RSA prime factors
Q_{seq}	total sequential oracle queries in the Parallelization model
Q_{step}	query counts for step $\in \{\text{trim, search, BU, total}\}$ in Trimmers, Searching, Bottom-Up, and the full attack
ρ	number of ℓ -bit Bottom-Up rounds after the initial modulus, $\rho = \lceil (n - d_0)/\ell \rceil$
r	target cluster selector in Bottom-Up
$r_{\min}, r_{\max}$	explicit lower and upper bounds on admissible r
R	local shorthand $R = rN/D$ in the Bottom-Up construction
R_q	number of round trips per query in the Parallelization model
s	query multiplier; in Bottom-Up often written as $s(v)$ for a guessed residue class
s_j	slice-specific multiplier in coverage-driven search
$s_{t,\varepsilon}$	trimmer multiplier for candidate divisor t and offset ε
S	lowest 46 bytes of the PreMasterSecret that are random; the additional 2 bytes of the PreMasterSecret are the version number constants
t	candidate trimmer divisor
τ	accumulated divisor in Trimmers, with $m = \tau \cdot m'$
u, u_l, u_h	accepting trimming multipliers; u_l and u_h are the smallest and largest accepted values in the local binary search
v	Bottom-Up residue class candidate modulo D
v^*	Bottom-Up correct residue class $x \bmod D$
w	Step 2a quotient with $m' \cdot s \in [wN + L, wN + U)$
x	generic working message in Bottom-Up and the appendix, instantiated with m'
X	currently known residue of x modulo D_{prev}
Y_v	real auxiliary accepting point $(\delta + k/D)x + rN/D$ in the batched rejection proof
z, z^*	batch selector for candidates $v = X + zD_{\text{prev}}$; z^* is the correct selector
$\mathbb{Z}_D^*$	units modulo D , i.e., the admissible residue-class guesses in the generic Bottom-Up theorem