Modular Roots from a Restricted Oracle: Faster Affine Roots and a 1024-bit Computation

Working Paper

Krishnamurthy (Dj) Dvijotham, Nadia Heninger*, Owen Jones, Pushmeet Kohli, Borislav Kozlovskii, Shrinu Kushagra, Chirag Pabbaraju*, Robert Stanforth, Abhradeep Thakurta, Emmanuel Thomé*, Nenad Tomašev, Adam Zsolt Wagner

Google DeepMind

Abstract

We study the computation of e-th roots modulo a fixed composite N in the presence of an oracle. After an offline phase in which the oracle returns e-th roots of chosen residues, the oracle is no longer present in an online phase in which roots of uniformly random targets must be computed. Joux, Naccache and Thomé introduced two versions of this problem: an *unrestricted* oracle, which accepts every residue, and an *affine* oracle, which accepts only residues $c + z$ for a fixed c and small integers z. The affine oracle models signing interfaces that apply a fixed format to the caller's data.

We first show that, for every fixed c, the format constraint that the affine oracle imposes stops mattering once the variable part z can be any integer up to roughly $\sqrt{N}$. Hence the PKCS#1 v1.5 format of `CKM_RSA_PKCS`, given more than half a block of data, or a 64-byte digest at 1024 bits, gives a problem no harder than the unrestricted oracle case, while a 32-byte digest leaves a more difficult affine problem.

For the affine problem, we lower the cost of computing one root, offline phase included, from $L_N[1/3, 1.8171]$ to $L_N[1/3, 1.5803]$; with a larger offline phase, of cost $L_N[1/3, 1.6373]$ and still far below factoring, the online cost per root approaches $L_N[1/3, 1.3867]$. The gains come from optimal multi-stage early abort, which we determine for every number of stages, and from a factor-base extension that uses only admissible inputs.

Finally, we report an improved experimental reproduction of the 1024-bit instance of Shea, Haller, Suhl, Heninger and Thomé, which relies on an unrestricted oracle. With the same algorithm, polynomials and bounds, the first root took 391 CPU core-years, 3.5 times less than the original computation. Two cooperating AI agents designed, ran and debugged this computation in under four days.

Important: Our results don't threaten deployed cryptosystems today that are based on 2048-bit and larger RSA. The main significance of these results is that they do sharpen our picture of what attacks older signature standards can resist, and how quickly AI-assisted research can move a field that usually advances slowly.

*Also affiliated with UC San Diego and Inria (Nadia Heninger), Stanford University (Chirag Pabbaraju) and Inria (Emmanuel Thomé).

1 Introduction

Let N be a composite integer and e a prime with $\gcd(e, |(\mathbb{Z}/N\mathbb{Z})^\times|) = 1$, so that $x \mapsto x^e$ permutes $(\mathbb{Z}/N\mathbb{Z})^\times$. Computing e-th roots modulo N without the factorization of N is the RSA problem, and the best known general method is to factor N with the number field sieve, in heuristic time $L_N[1/3, c_{\mathrm{GNFS}}]$, where

$$L_N[a, x] := \exp\left((x + o(1))(\ln N)^a (\ln \ln N)^{1-a}\right) \qquad \text{and} \qquad c_{\mathrm{GNFS}} := (64/9)^{1/3} \approx 1.9230.$$

Joux, Naccache and Thomé [JNT07] asked how much easier the problem becomes after temporary access to an oracle for e-th roots. In their model, an algorithm first runs an *offline phase* in which it may query an oracle that returns e-th roots modulo N; the oracle is then withdrawn, and in the *online phase* the algorithm receives uniformly random targets $t \in (\mathbb{Z}/N\mathbb{Z})^\times$ and must output $t^{1/e} \bmod N$. The queries are chosen before the targets are known, so the model is non-adaptive. Two versions of the oracle have been studied:

- an *unrestricted* oracle, which accepts every residue; and
- an *affine* oracle, which, for a fixed residue c and a bound Z, accepts only the residues $c + z$ with $z \in \mathbb{Z}$ and $|z| \le Z$.

Following [JNT07], we call the problem with the affine oracle the *affine modular root* (AMR) problem. Its oracle models a party that applies the secret root map only to inputs of a fixed format: a fixed part, represented by c, and a variable part z chosen by the caller. This is how keys held in cryptographic devices are typically used: a signing interface pads the caller's data in a standard format before applying the map. Figure 1 illustrates the model.

For the unrestricted oracle, the algorithm of [JNT07] runs in time $L_N[1/3, c_{\mathrm{SNFS}}]$ offline and $L_N[1/3, c_{\mathrm{SNFS}}]$ per root, with $L_N[1/3, (4/9)^{1/3}]$ queries, where $c_{\mathrm{SNFS}} := (32/9)^{1/3} \approx 1.5263$ is the constant of the special number field sieve (see the full version of this paper). The offline exponent 1.577 stated in [JNT07] belongs to an asymptotically suboptimal variant with a larger factor base, which behaves better in practice at moderate sizes; its online exponent is also c_{SNFS}. Shea, Haller, Suhl, Heninger and Thomé [SHS+26] recently carried out this algorithm for a 1024-bit modulus, with about 1,200 core-years offline, 4.07×10^9 queries and 180 core-years for the root of one target. For the affine oracle, the algorithm of [JNT07] costs $L_N[1/3, 6^{1/3}] \approx L_N[1/3, 1.8171]$ per root, and [SHS+26] expect it to be much less practical than the unrestricted algorithm.

Because the offline phase is paid once and the online phase once per target, we measure an algorithm by its *amortized cost per root*: for M targets, the offline cost divided by M, plus the online cost of one target. All complexity statements about the number field sieve and the AMR algorithm assume the standard heuristics of number field sieve analyses (Section 2.1), and we write $L[x] := L_N[1/3, x]$.

1.1 Results

The paper has three results. The first shows which input formats pose a problem no harder than the unrestricted one; the second improves the algorithm for the affine problem; the third is a complete computation for the unrestricted problem at 1024 bits.

When the format stops mattering. The length of the variable part decides whether a simple reduction makes the affine oracle as strong as the unrestricted one. If $c + z \equiv \rho(c + z') \pmod N$, the root of ρ is the quotient of the roots of two admissible inputs, and writing a residue as such a ratio is a closest-vector problem in a two-dimensional lattice of determinant N.

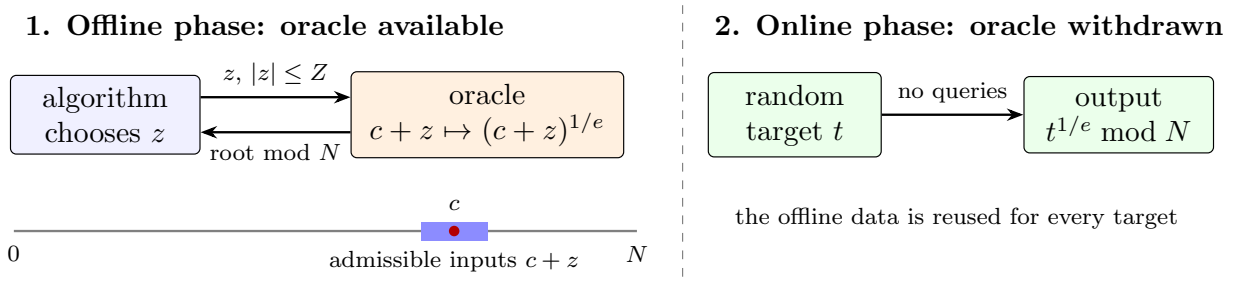


Figure 1: The oracle model. In the offline phase the algorithm chooses offsets z and receives $(c+z)^{1/e} \bmod N$; the affine oracle accepts only $|z| \le Z$, so all its inputs lie in a short window around the fixed residue c, whereas the unrestricted oracle accepts every residue. After the oracle is withdrawn, the algorithm computes the roots of random targets t without further queries.

Theorem (Theorem 4.1, Corollary 4.2 and Proposition 4.3)**.** *For every fixed residue c, an affine oracle whose window holds $O(N^{1/2})$ offsets simulates the unrestricted oracle with two queries per query, all made before the target is known, after a constant number of re-randomizations per query on average. In practice, this holds down to $N^{1/2}$ offsets; below $N^{1/2}$, a counting argument bounds the number of re-randomizations from below by a quantity that quadruples with each bit removed from the window.*

All of these statements except the one about practice are proved, without heuristic assumptions. In our measurements at 1024 bits, a one-sided window of 2^{512} offsets, that is, a variable part of exactly half the bits of a random modulus, needs 1.26 re-randomizations per query on average (Figure 4). For standard interfaces this has a sharp consequence (Section 4.2 and Figure 5). The PKCS#1 v1.5 format of the PKCS#11 mechanism `CKM_RSA_PKCS` accepts up to $n_B - 11$ bytes of data, where n_B is the length of N in bytes; with more than about $n_B/2$ bytes of data, or with a 64-byte digest at 1024 bits, the format does not make the problem harder than the raw map `CKM_RSA_X_509`, for which [JNT07, SHS+26] and our computation apply. Shorter variable parts, such as a 32-byte digest, are out of reach of this reduction and leave an affine problem.

A faster algorithm for the affine problem. The algorithm of [JNT07] pays its bottleneck, the initialization of the descent, for every target, so its amortized cost per root is $L[1.8171]$ for any number of targets.

Theorem (Theorem 3.1)**.** *For every fixed residue c and every offset bound $Z = N^\theta$ with $\theta > 0$, the AMR problem can be solved with $L[0.9602]$ queries, offline time $L[1.5803]$ and online time $L[1.5803]$ per root. With offline time $L[1.5930]$ the online time per root is $L[c_{\mathrm{SNFS}}]$, and with offline time $L[1.6373]$ it can be made arbitrarily close to $L[(8/3)^{1/3}] = L[1.3867]$. For every choice of parameters, the offline exponent of the method is at least c_{SNFS}.*

The trade-off ends at offline time $L[1.6373]$, far below the $L[c_{\mathrm{GNFS}}]$ of factoring N, because the online exponent cannot drop below the limit $(8/3)^{1/3}$ of the initialization. The main tool is a statement about smoothness testing. One must find, among random integers of size N^y, one whose prime factors are all below $L_N[2/3, x]$; testing each candidate with one run of the elliptic curve method (ECM) finds one in time $L[(3y)^{1/3}]$, which is $L[6^{1/3}]$ for the integers of size N^2 that arise in [JNT07]. An *early-abort* test with s stages discards a candidate as soon as the smooth part found so far is too small.

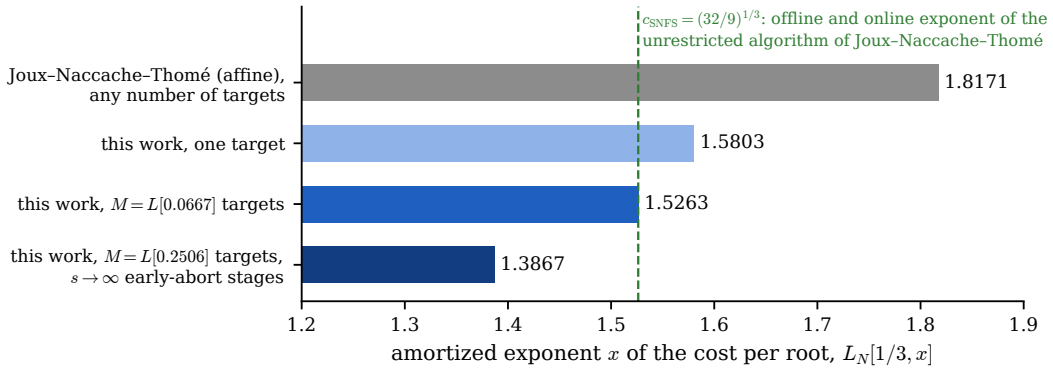


Figure 2: Amortized cost per root in the AMR model, as the exponent x of $L[x] = L_N[1/3, x]$. Batches of $L[0.0667]$ and $L[0.2506]$ targets share one offline phase, of exponents 1.5930 and 1.6373. The dashed line marks c_{SNFS}, the offline and online exponent of the algorithm of [JNT07] for an oracle that accepts every residue; the offline exponent of our method is never below it.

Theorem (see the full version)**.** *For every $s \geq 1$, the optimal thresholds of s-stage early-abort testing give cost $L[(3(1 - C_s)^2 y)^{1/3}]$, where C_s is the largest left Riemann sum of r^2 on s intervals of $[0, 1]$; C_s increases to $1/3$, so the cost decreases to $L[(4y/3)^{1/3}]$, and no s-stage procedure does better.*

A 1024-bit computation. Complete computations at cryptographic sizes are possible today for the unrestricted algorithm. We implemented it in C++ on top of CADO-NFS [CAD], with $e = 65537$. On the modulus, polynomials and factor-base bounds of [SHS+26], the first root took 391 CPU core-years, 318 offline and 73 online, against 1,380 (1,200 and 180) reported for their computation; factoring the modulus is estimated at 500,000 to 1,000,000 core-years. Both computations run the same algorithm on the same polynomials, so the factor of 3.5 is one of implementation and execution, not of algorithms (Section 5). By the first result, the same computation applies to the fixed formats above the threshold (with the minor difference that the number of queries is doubled).

What stands out in the 1024-bit computation

- 3.5 *times less CPU time on the same instance.* The first root took 391 CPU core-years against 1,380 for [SHS+26], with the same algorithm, polynomials and factor-base bounds: 2.7 and 4.0 times less in the two sieving stages, 2.2 times less in the descent and about 6 times less in the root extraction. The saving comes mainly from the code rather than from the hardware: on the same machine, the optimized siever is 1.8 to 4.4 times faster than CADO-NFS and finds the same relations, or, when it stops at the first relation, relations for the same prime ideals; the descent and the root step use cheaper strategies within the same algorithm (Section 5.3).
- *Done by AI agents with almost no human input.* The computation was designed, implemented, optimized, run and debugged by two cooperating AI agents: a supervisor (Model A), which wrote the design, set checkpoints and audited every result, and an executor (Model B), which wrote the code, ran the distributed jobs and fixed failures. The human researcher sent about twenty short messages, most of them approvals. The agents wrote about 33,500 lines of new code, ran 336 distributed experiments, and resolved during the run the problems that a smaller rehearsal could not expose, including descent nodes of 82 to 85 bits that the descent mode of the siever abandoned and a 32-bit index overflow that made the first computed root wrong (Table 4).
- *Calendar time.* Running on up to 300,000 CPU cores and 16 GPUs, the agents completed the computation in 3.2 days from the first full-scale sieving job, against about five months for [SHS+26]; this difference reflects mainly parallelism.

Section 6 describes the agent setup.

Contributions.

1. *The square-root threshold.* For every fixed c, the reduction to the unrestricted oracle with two queries per query is proved down to $O(N^{1/2})$ offsets, works in practice down to $N^{1/2}$ offsets, and requires exponentially many draws below $N^{1/2}$ (Theorem 4.1, Corollary 4.2, Proposition 4.3, and measurements at 1024 bits).
2. *Consequences for standard interfaces.* A classification of the PKCS#11 signing mechanisms by the result that applies to each: raw map and long data (unrestricted), 64-byte digest at 1024 bits (unrestricted, at the threshold), 32-byte digest (affine), digest computed inside (outside both models) (Section 4.2).
3. *Optimal multi-stage early abort.* The optimal thresholds and cost of s-stage early-abort smoothness testing, for every s, improving the stage-by-stage parameters of Barbulescu [Bar13] from three stages on (see the full version).
4. *A faster AMR algorithm.* Amortized cost per root $L[1.5803]$ for one target, improving the bound of $L[1.8171]$ obtained by [JNT07]; an offline–online trade-off down to $L[c_{\mathrm{SNFS}}]$ and arbitrarily close to $L[1.3867]$ online; and a lower bound c_{SNFS} on the offline exponent of the method (Theorem 3.1).
5. *A complete 1024-bit computation* of the unrestricted algorithm, 3.5 times cheaper in CPU time than the previous one on the same instance, and a description of the agent system that designed and ran it (Sections 5 and 6).

1.2 Technical overview

The threshold. For a residue r, draw a random unit s and put $\rho = rs^e$. The pairs (x, y) with $x \equiv \rho y \pmod N$ form a lattice Λ_ρ of determinant N, and $c + z \equiv \rho(c + z')$ says that (z, z') lies in a fixed coset of it. If Λ_ρ has no vector shorter than $N^{1/2}/4$, its covering radius is bounded so

that Lagrange–Gauss reduction followed by Babai's nearest-plane step finds a coset point with both coordinates below $2.08N^{1/2}$, whereas a short vector (x, y) determines $\rho \equiv xy^{-1}$, which happens for few ρ. Conversely, the ratios of W admissible inputs take at most $2W^2$ values, which bounds the success rate of any method of this kind by $2W^2/\varphi(N)$.

The AMR algorithm. When the window is short, the algorithm works in two number fields of degree $d \geq 2$ whose defining polynomials share the root c modulo N; their elements $\alpha_i + z$ map to the admissible inputs $c + z$. Three costs govern the algorithm: the initialization, which multi-stage early abort lowers from $L[6^{1/3}]$ towards $L[(8/3)^{1/3}]$; the factor-base extension, offline; and the last level of the descent, online. A larger factor base makes the extension more expensive and the last level cheaper, and balancing the two gives 1.5803 (Section 3).

1.3 Related work

Oracle-assisted roots. Joux, Naccache and Thomé [JNT07] introduced the non-adaptive problem with both oracles. For the unrestricted oracle they gave two different complexity estimates. The first one has matching offline and online costs $L[c_{\mathrm{SNFS}}]$ with $L[(4/9)^{1/3}]$ queries, and is described as an asymptotic result only (see the full version). An alternative, presumed more practical setting is proposed by [JNT07] with offline cost $L[1.577]$, $L[(8/9)^{1/3}]$ queries, and unchanged online cost $L[c_{\mathrm{SNFS}}]$. For the affine oracle, [JNT07] suggests a descent-based algorithm of cost $L[6^{1/3}]$ whose bottleneck is the initialization of the descent.

Shea, Haller, Suhl, Heninger and Thomé [SHS+26] implemented the unrestricted algorithm with a two-polynomial number field sieve and computed a root for a 1024-bit modulus; they used an oracle that accepts every residue and discuss the affine version only asymptotically [SHS+26, Section 5.2].

Descent algorithms. In discrete logarithm computations, a procedure known as the descent is used to link an arbitrary target to the precomputed virtual logarithms. That procedure has a long history: Coppersmith [Cop84], Gordon [Gor93], Schirokauer [Sch93], Semaev [Sem98], Adleman and Huang [AH99], and Joux and Lercier [JL03] all contributed to the development of the descent, and the version of Commeine and Semaev [CS06] was the first with a correct proof of a cheap online phase in the number field sieve setting. Both ends of the descent, namely the initialization (smoothing) step and the last level of the descent, deserve attention in our analysis.

Smoothness testing with early abort. Early abort was introduced by Pomerance [Pom82] for the analysis of factoring algorithms. Barbulescu [Bar13, Chapter 4] applied multi-stage early abort with ECM to the smoothing problem that starts an individual-logarithm descent in the number field sieve, and lowered its exponent from $3^{1/3} \approx 1.442$ to about 1.232; this is the closest prior work to our early-abort theorem. Early abort is also used in the batch number field sieve analysis of Bernstein and Lange [BL14]. Our early-abort theorem determines the optimal thresholds for every number of stages: it agrees with [Bar13] for two stages and improves on it from three stages on, with limit $(4/3)^{1/3} \approx 1.1006$ instead of about 1.232 for integers of size N (see the full version). To our knowledge, multi-stage early abort had not been applied to the initialization of the AMR descent of [JNT07], where the single-stage cost $6^{1/3} \approx 1.8171$ had been the bottleneck.

1.4 Organization

Section 2 fixes notation, the heuristics and the two problems. Section 3 describes the AMR algorithm, explains what its analysis shows, and states the main theorem. Section 4 presents the square-root threshold, quantifies the method below it, and maps the results onto the PKCS#11 interfaces. Section 5 reports the 1024-bit computation and compares it with [SHS+26], and Section 6 describes the agent system that ran it. Section 7 discusses limitations and open problems. The full version of this paper gives the AMR algorithm in full, the proofs omitted here, the stages of the implementation and the parameters of the computation.

2 Preliminaries

Scales. Throughout, N is the fixed modulus and e a prime with $\gcd(e, \varphi(N)) = 1$, where $\varphi(N) := |(\mathbb{Z}/N\mathbb{Z})^\times|$. We write $L[x] := L_N[1/3, x]$. Since $L[a]L[b] = L[a+b]$ and $L[a] + L[b] = L[\max\{a, b\}]$, cost statements reduce to comparisons of exponents, and factors of size $(\ln N)^{O(1)}$ are $L[o(1)]$. The degree of the number fields is $d = (\delta + o(1))(\ln N/\ln\ln N)^{1/3}$ for a constant $\delta > 0$, so that

$$N^{1/d} = L_N[2/3, 1/\delta], \qquad L[x]^d = L_N[2/3, x\delta], \qquad L_N[2/3, \zeta]^d = N^{\zeta\delta + o(1)}. \tag{1}$$

An integer is *B-smooth* if none of its prime factors exceeds B. We write $\mathbb{F}_e := \mathbb{Z}/e\mathbb{Z}$.

Smoothness. By the theorem of Canfield, Erdős and Pomerance [CEP83], a uniformly random positive integer below X is B-smooth with probability $v^{-v(1+o(1))}$, where $v = \ln X/\ln B$, provided B is not too small relative to X. In the two regimes used in this paper this gives

$$\begin{aligned} \Pr\left[L[g]\text{-smooth}\right] &= L\left[-\frac{s}{3g}\right] \quad \text{for } X = L_N[2/3, s], \\ \Pr\left[L_N[2/3, x]\text{-smooth}\right] &= L\left[-\frac{\nu}{3x}\right] \quad \text{for } X = N^\nu. \end{aligned} \tag{2}$$

Polynomials and norms. For $f = \sum_{j=0}^{d} f_j x^j \in \mathbb{Z}[x]$ irreducible with root α, and coprime integers a, b, the homogenization $F(a, b) := b^d f(a/b)$ satisfies $F(a, b) = \mathrm{Res}(f, a - bx) = f_d \,\mathrm{N}_{\mathbb{Q}(\alpha)/\mathbb{Q}}(a - b\alpha)$. A prime ideal of degree one of $\mathbb{Z}[\alpha]$ is a pair $\mathfrak{q} = \langle q, \alpha - r\rangle$ with q prime and $f(r) \equiv 0 \pmod q$, and $\mathrm{N}\,\mathfrak{q} = q$.

2.1 Heuristics

Like the standard analyses of the number field sieve [LLMP90, BLP93, JNT07], ours rests on unproven heuristics, which all running-time statements about the number field sieve and the AMR algorithm assume.

(H1) *Smoothness.* The integers to be tested for smoothness (values of polynomials, norms of algebraic integers, and reduced residues) are smooth with the probabilities (2) of random integers of the same size, independently across the two sides of a relation; a candidate algebraic integer is divisible by no prime ideal of degree at least two with probability bounded away from zero.

(H2) *Linear algebra.* Time is measured in the unit-cost RAM model. Sieving a region of $L[x]$ points with a factor base of at most $L[x]$ primes costs $L[x]$, and a sparse $D \times D$ system over $\mathbb{F}_e$ with $L[o(1)]$ non-zero entries per row is solved by (block) Wiedemann in time $D^2 L[o(1)]$.

(H3) *e-th powers.* An algebraic integer whose principal ideal is the e-th power of an ideal, and on which $O(\ln N)$ e-th power characters vanish, is an e-th power in its number field [BLP93, JNT07].

(H4) *Reduced lattices.* A lattice of dimension $k = O(d)$ and covolume V in a weighted norm has an LLL-reduced basis [LLL82] whose vectors have weighted size $V^{1/k}2^{O(k)} = V^{1/k}N^{o(1)}$, and small combinations of basis vectors have norms that behave like random integers of the same size with respect to H1; the reduced vectors used as defining polynomials are irreducible and of exact degree d.

(H5) *Elliptic curve method.* Finding all prime factors $p \le B_{\text{ecm}}$ of an integer with ECM [Len87] takes time $L_{B_{\text{ecm}}}[1/2, \sqrt{2}]$. For $B_{\text{ecm}} \le L_N[2/3, \zeta]$ this is at most $L[\sqrt{4\zeta/3}]$, and for $B_{\text{ecm}} = L[x]$ it is $L[o(1)]$.

2.2 The problems

Definition 2.1 (Non-adaptive root problems [JNT07])**.** Let N and e be as above. An *oracle* is a partial map $\mathcal{O}$ on $\mathbb{Z}/N\mathbb{Z}$ with $\mathcal{O}(w) = w^{1/e} \bmod N$ wherever it is defined. In the *offline phase*, an algorithm receives the public data and may query $\mathcal{O}$. In the *online phase*, $\mathcal{O}$ is no longer accessible; the algorithm receives targets $t_1, \dots, t_M$, drawn independently and uniformly from $(\mathbb{Z}/N\mathbb{Z})^\times$, and must output $t_m^{1/e} \bmod N$ for every m. We consider two oracles:

(i) the *unrestricted* oracle $\mathcal{O}_{\text{all}}$, defined on all of $\mathbb{Z}/N\mathbb{Z}$;

(ii) for a residue $c \in \mathbb{Z}/N\mathbb{Z}$ and a bound $Z \ge 1$, the *affine* oracle $\mathcal{O}_{c,Z}$, defined on $\{c + z : z \in \mathbb{Z},\ |z| \le Z\}$; the public data are (N, e, c, Z), and c is given, not chosen.

The *AMR problem* is the problem with oracle $\mathcal{O}_{c,Z}$. We write $Z = N^\theta$ when the bound is a power of N.

Since e is odd, $(-w)^{1/e} = -w^{1/e}$, so an oracle defined at w also gives the root of $-w$; and since the root map is multiplicative, roots of products are products of roots. Every query to $\mathcal{O}_{c,Z}$ is also a query to $\mathcal{O}_{\text{all}}$, so the unrestricted problem is no harder than the AMR problem.

Cost measure. For M targets, an algorithm with offline cost P and online cost R per target has *amortized cost per root* $P/M + R$. With $P = L[\Pi]$, $R = L[\Omega]$ and $M = L[\mu]$, this is $L[\max\{\Pi - \mu, \Omega\}]$. In Section 5, P and R are measured CPU times; factoring N is the reference point, since after it every root costs one exponentiation.

Remark 2.2 (Formats with a fixed part). Suppose the inputs to the root map are byte strings of the length of N in which a prefix is fixed and only the last k bits are chosen by the caller. Then the admissible inputs are $\{c + z : 0 \le z < 2^k\}$ for the integer c encoded by the fixed prefix followed by k zero bits: an affine oracle with a one-sided window of 2^k offsets. Section 4 shows that such an oracle is as strong as the unrestricted one from about $k = \frac{1}{2}\log_2 N$ on, for every prefix, and applies this to standard interfaces.

3 The AMR algorithm

This section describes our algorithm for the AMR problem, explains informally what its analysis shows, and states the main theorem. The algorithm follows the template of [JNT07, Section 4.3]. An offline phase collects smooth relations on a core factor base of prime ideals and on an extended

factor base, querying the oracle on the corresponding elements; an online phase rewrites each target in terms of the extended factor base, first by an *initialization* and then by a *descent*, reduces to the core factor base, and solves the core linear system over $\mathbb{F}_e$ to form an algebraic e-th power and extract its root, without the oracle. Two changes make it faster: the initialization uses multi-stage early abort with optimal thresholds, and the factor base is extended offline by larger prime ideals, each linked to the oracle by one element whose image the oracle accepts. The full version of this paper gives the algorithm in full, the analysis and all proofs.

The algorithm in brief. Take two irreducible polynomials f_1, f_2 with $f_i(c) \equiv 0 \pmod N$, of degree d proportional to $(\ln N/\ln\ln N)^{1/3}$ and chosen as short vectors of a skewed lattice,[1] and let $K_i = \mathbb{Q}(\alpha_i)$ with $f_i(\alpha_i) = 0$. The map $\alpha_i \mapsto c$ sends the element $\alpha_i + z$ of K_i to $c + z$, which the oracle accepts, and sends $\psi(\alpha_1)$ and $\psi(\alpha_2)$ to the same residue $\psi(c)$ for every $\psi \in \mathbb{Z}[X]$. Offline, the algorithm collects elements $\alpha_i + z$ with smooth norms and queries the roots of their images, covering all prime ideals of norm up to a bound $B' = L[\beta']$ (a core factor base up to $B = L[\beta]$ and an extension on $(B, B']$). Online, for each target, it writes the target as a product of medium-sized prime ideals, eliminates those above B' by polynomials ψ for which $\psi(\alpha_1)$ and $\psi(\alpha_2)$ are smooth in both fields, eliminates the ideals in $(B, B']$ by back-substitution, solves the $L[\beta] \times L[\beta]$ sparse linear system over $\mathbb{F}_e$, and extracts the algebraic e-th roots.

What the analysis shows. Three costs determine the running time, and the analysis computes each of them.

- *The factor-base extension* must give every prime ideal of norm up to B' an element $\alpha_i + z$ with smooth norm and an offset that the oracle accepts. The elements $\alpha_i + z$ of a prime ideal of norm q have offsets spaced q apart, so the offsets grow with q; we choose the skewness of f_1, f_2 equal to the largest offset needed. The skewness enlarges the norms of the one-dimensional elements $\alpha_i + z$ but cancels from the norms of two-dimensional elements $a - b\alpha_i$, and the extension costs $L[\sigma_0(\delta, \beta')]$ with $\sigma_0(\delta, \beta') = (6\beta'^2 + 2/\delta)/(6\beta' - \delta)$ (see the full version).
- *The initialization of the descent* must find a smooth integer among random integers of size N and then N^2. Testing every candidate with one run of the elliptic curve method (ECM), as in [JNT07], costs $L[6^{1/3}] \approx L[1.8171]$ per target. An s-stage early-abort test discards most candidates in cheap early stages. With the optimal thresholds, which come from the largest left Riemann sum of $r \mapsto r^2$ on s intervals of $[0, 1]$, the cost tends to $L[(8/3)^{1/3}] \approx L[1.3867]$ as s grows, because these sums tend to $\int_0^1 r^2\, dr = 1/3$. For one target, three stages already make the initialization cheaper than the other two costs, and four stages bring it below c_{SNFS} (see the full version).
- *The last level of the descent* eliminates prime ideals of norm just above B' by relations that must be smooth in two fields of degree d. It costs $L[\omega(\delta, \beta')]$ with $\omega(\delta, \beta') = (2/\delta + \delta\beta')/(3\beta' - \delta)$ (see the full version), much more than in the unrestricted setting, where one of the two polynomials is linear.

A larger extended bound B' makes the extension more expensive and the last level cheaper: ω falls with β', and σ_0 grows with β' beyond its minimum, which lies near $\beta' = 0.77$. Balancing the two gives the main theorem.

[1] We assume here that the lattice is generic, so that we can bound two short vectors. If an exceptionally short vector exists, which means c is a root modulo N of a polynomial of degree at most d with abnormally small coefficients, we need a minor adjustment: by shifting the query window by a suitable offset, the result stands (see the full version).

Table 1: Heuristic costs of computing e-th roots modulo N after an offline phase of oracle access, as exponents x of $L[x] = L_N[1/3, x]$. The amortized exponent for $M = L[\mu]$ targets is $\max\{\text{offline} - \mu, \text{online}\}$. For the algorithm of [JNT07, Section 4.3] only the total $6^{1/3}$ is listed; it bounds the number of queries and the offline cost.

Oracle	Algorithm	Queries	Offline	Online	Amortized, $M = L[\mu]$
unrestricted	[JNT07]	0.7631	1.5263	1.5263	1.5263 for all $\mu \geq 0$
affine	[JNT07, Section 4.3]	≤ 1.8171	≤ 1.8171	1.8171	1.8171
affine	Theorem 3.1(a)	0.9602	1.5803	1.5803	1.5803 at $\mu = 0$
affine	Theorem 3.1(b)	0.9871	1.5930	1.5263	1.5263 at $\mu = 0.0667$
affine	Thm. 3.1(b), $s \to \infty$	1.0704	1.6373	1.3867	1.3867 at $\mu = 0.2506$

Theorem 3.1 (Main AMR theorem)**.** *Assume H1–H5, and fix $\theta > 0$. For every residue $c \in \mathbb{Z}/N\mathbb{Z}$, the AMR problem with offset bound $Z = N^\theta$ can be solved as follows.*

(a) One target: *with three early-abort stages, $L[0.9602]$ queries, offline time $L[1.5803]$ and online time $L[1.5803]$.*

(b) Many targets: *with four early-abort stages, $L[0.9871]$ queries and offline time $L[1.5930]$, online time $L[c_{\mathrm{SNFS}}] = L[1.5263]$ per root; and with $L[1.0704]$ queries and offline time $L[1.6373]$, online time per root arbitrarily close to $L[(8/3)^{1/3}] = L[1.3867]$ as the number of stages grows.*

No choice of the parameters of the method gives offline time below $L[c_{\mathrm{SNFS}}]$.

Idea of the proof. The offline cost is the extension cost (which dominates the core relation collection), and the online cost is the largest of the initialization cost, the descent cost and the $L[2\beta]$ core linear algebra (which is non-dominating for suitable β). Since ω decreases with β' while σ_0 increases beyond its minimum, the single-target optimum lies on the curve $\sigma_0 = \omega$, and minimizing along it gives the single-target exponent $c_{\mathrm{AMR}} \approx 1.5803$ at $(\delta, \beta') \approx (1.0205, 0.9602)$, where three early-abort stages already make the initialization cheaper. Raising β' from there lowers ω about 4.6 times faster than it raises σ_0, which gives part (b) until ω meets the limit $(8/3)^{1/3}$ of the initialization. The last statement is a discriminant computation: $\sigma_0(\delta, \beta') \geq c_{\mathrm{SNFS}}$ for all $\delta > 0$ and $\beta' > \delta/6$. The full version states the theorem with all its parameters and gives the proofs.

The trade-off ends at offline time $L[1.6373]$, far below the $L[c_{\mathrm{GNFS}}]$ of factoring N: beyond it, the online exponent stays at the limit $(8/3)^{1/3}$ of the initialization, so a larger offline budget buys nothing. Table 1 compares the exponents with previous algorithms, and Figure 3 shows the trade-off. None of these algorithms computes more roots than it made queries at less than the cost of factoring (see the full version); the gains are gains in the cost per root for a moderate number of targets. It is worth noting that the factor-base extension exists in both the unrestricted oracle case of [JNT07] and the affine oracle case that we study here in more depth. There is a key difference, however: here, the bound $B' = L[\beta']$ is computed as an optimum that satisfies precise constraints, unlike what is done in [JNT07] in the unrestricted setting, where the $L[1.577]$ complexity is driven by the arbitrary choice $\beta' = (8/9)^{1/3}$ (see the full version).

4 When is the affine oracle as strong as the unrestricted one?

The affine oracle models a party that applies the secret root map only to inputs of a fixed format: a fixed part, represented by c, and a variable part z chosen by the caller. Section 3 treated short

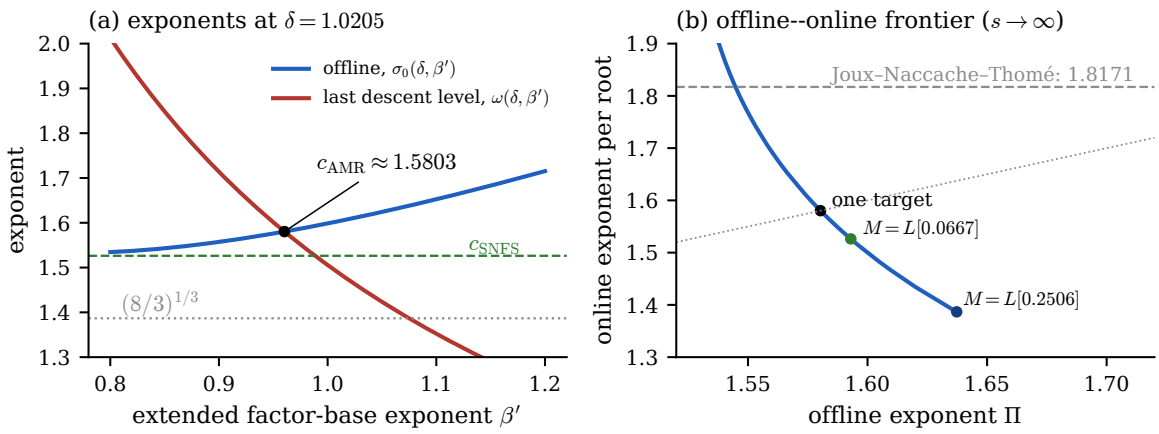


Figure 3: (a) The offline exponent $\sigma_0(\delta, \beta')$ of the factor-base extension and the exponent $\omega(\delta, \beta')$ of the last descent level, at the optimal δ of Theorem 3.1(a); they cross at $c_{\mathrm{AMR}} \approx 1.5803$. (b) The smallest online exponent achievable with a given offline exponent, as the number of early-abort stages grows (a numerical minimization over δ and β' subject to the constraints of the main theorem with all its parameters, stated in the full version). The marked points are the three cases of Theorem 3.1; the dotted diagonal marks equal offline and online exponents. The frontier ends at offline exponent 1.6373, where the online exponent reaches the limit $(8/3)^{1/3}$ of the initialization.

variable parts. This section determines the length of the variable part at which the format stops mattering, and then maps this answer onto standard signing interfaces. Section 4.1 proves that a window of $O(N^{1/2})$ offsets suffices for every c, shows that the same method works in practice down to $N^{1/2}$ offsets, and shows that it cannot go much below $N^{1/2}$. Section 4.2 applies these results to the mechanisms of PKCS#11 [OAS15].

4.1 The square-root threshold

The idea is elementary. If $c + z \equiv \rho(c + z') \pmod N$, the root of ρ is the quotient of the roots of the two admissible inputs $c + z$ and $c + z'$. Writing a prescribed residue as such a ratio is a closest-vector problem in a two-dimensional lattice of determinant N, whose solutions have offsets of size about $N^{1/2}$. Re-randomizing the residue, $\rho = rs^e$ for a random unit s, makes the lattice random, and $r^{1/e} = \rho^{1/e} s^{-1}$. In this section we assume $N = pq$ with odd primes $p < q < 2p$, as for RSA moduli.

Theorem 4.1 (The affine oracle above the square root)**.** *Let $c \in \mathbb{Z}/N\mathbb{Z}$ be arbitrary and $Z \geq 2.08N^{1/2}$. Let $\mathcal{A}$ be an algorithm that makes Q queries to $\mathcal{O}_{\mathrm{all}}$, all before the target is known. Then $\mathcal{A}$ can be run with $\mathcal{O}_{c,Z}$ instead, using $2Q$ queries, all before the target is known, and in addition an expected number of at most $2Q$ reductions of two-dimensional lattices with entries at most N. The same holds for the one-sided window $\{c + z : 0 \leq z < W\}$ with $W \geq 4.16N^{1/2} + 2$. Should the simulation meet a nonzero residue that is not a unit, it finds a nontrivial factor of N.*

Idea of the proof. The ratio lattice $\Lambda_\rho = \{(x, y) \in \mathbb{Z}^2 : x \equiv \rho y \pmod N\}$ has determinant N, so its covering radius is governed by $N/\lambda_1(\Lambda_\rho)$: once $\lambda_1(\Lambda_\rho) \geq N^{1/2}/4$, Lagrange–Gauss reduction and Babai's nearest-plane step reach any target point within distance $2.08N^{1/2}$, which gives the two offsets. A lattice with a shorter vector (x, y) determines $\rho \equiv xy^{-1}$, so at most $N/4$ residues have one, and a random re-randomization avoids them with probability at least $1/2$. The proof is in the full version; Theorem 4.1 makes no heuristic assumption.

Theorem 4.1 is not the limit of the method. Requiring a smaller covering radius—that is, a larger minimum $\lambda_1(\Lambda_\rho)$—lets Babai's step reach the target within a smaller distance, so a smaller window suffices; fewer re-randomizations meet this requirement, but still a constant fraction of them.

Corollary 4.2 (Smaller windows)**.** *Let c be any fixed residue, and let the window hold more than $2.3N^{1/2}$ consecutive offsets. Then, for all sufficiently large N, each re-randomization in Theorem 4.1 succeeds with probability bounded below by a positive constant that depends only on the ratio of the window size to $N^{1/2}$ and tends to 1 as this ratio grows. Each query of the unrestricted algorithm still costs two queries to the affine oracle.*

The window of the corollary is 1.2 bits above half the bits of N; the full version gives the explicit bound and its proof. Figure 4 shows that the method works in practice even at exactly half the bits: for random 1024-bit moduli and the one-sided window of 2^{512} offsets, which is the window of a 64-byte variable part, 79.6% of the draws succeed, so a query costs 1.26 draws on average. A failed draw costs one two-dimensional reduction and no query.

Why re-randomization stops near half the bits. Below the threshold, the method is limited by a count, not by the lattice reduction.

Proposition 4.3 (Limit of re-randomization)**.** *Let c be any fixed residue, and let the window hold W offsets. A random residue is, up to sign, the ratio of two admissible inputs with probability at most $(2 + o(1))W^2/N$. Hence every method that re-randomizes until it finds such a ratio needs at least about $N/(2W^2)$ draws per query: below half the bits of N, every bit removed from the window multiplies this number by four.*

The proof is a two-line count (see the full version); the sign is allowed because $(-w)^{1/e} = -w^{1/e}$. The measurements of Figure 4 follow the count: from 2^{511} offsets down, the measured success rate per draw is within 10% of W^2/N (the measured procedure, like that of Theorem 4.1, uses only ρ itself, not $-\rho$), and it falls by a factor of about four per bit, from 31.7% at 2^{511} to 0.53% at 2^{508}. The proposition bounds only methods that write each queried residue as one ratio of two admissible inputs; it does not rule out other simulations such as multiplicative combinations of three queries or more. For short variable parts, the relevant algorithm is the one of Section 3, which uses the admissible inputs as elements $\alpha_i + z$ of number fields of degree at least two.

4.2 Standard interfaces

The PKCS#11 standard [OAS15] specifies how applications use keys held in cryptographic devices (such as hardware security modules). Each of its mechanisms fixes how the block to which the root map is applied is formed from the caller's input, and therefore determines a fixed part c and a window of offsets. Write n_B for the length of N in bytes; $n_B = 128$ for a 1024-bit modulus. Figure 5 summarizes the cases for a 1024-bit modulus.

(i) Raw map. The mechanism `CKM_RSA_X_509` applies the root map to any residue. This is the unrestricted oracle, the setting of [JNT07, SHS+26] and of the computation of Section 5.

(ii) Fixed format, long data. The mechanism `CKM_RSA_PKCS` places the caller's data, of any length $\ell \le n_B - 11$ bytes, at the end of the fixed-format block of PKCS#1 v1.5 [MKJR16]. For each fixed ℓ, the blocks are the residues $c_\ell + z$ with $0 \le z < 2^{8\ell}$, where c_ℓ depends only on ℓ and

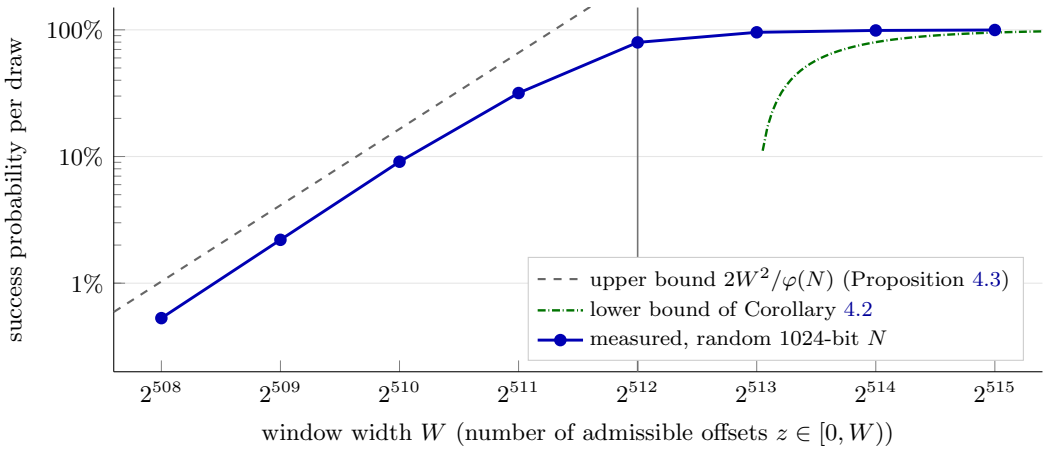


Figure 4: Success probability of one re-randomization at 1024 bits, for the one-sided window $\{c+z : 0 \le z < W\}$, measured over 10^4 draws per window size; each draw takes a fresh random odd N uniform in $[2^{1023}, 2^{1024})$ (mean $\log_2 N = 1023.6$) and a fresh c, with Lagrange–Gauss reduction, exact rounding and a search of the 3×3 lattice points around the rounded solution. The vertical line marks $W = 2^{512}$, the window of a 64-byte variable part. The dashed line is the upper bound $2W^2/\varphi(N)$ of Proposition 4.3 and the dash-dotted line the lower bound of Corollary 4.2, both for $\log_2 N = 1023.6$; Theorem 4.1 applies from $W = 4.16N^{1/2} + 2 \approx 2^{513.9}$ on. The explicit forms of both bounds are proved in the full version.

n_B: an affine oracle with a one-sided window of $W = 2^{8\ell}$ offsets. Theorem 4.1 applies as soon as $8\ell \ge \frac{1}{2}\log_2 N + 2.1$, that is, for $\ell \ge 65$ at 1024 bits, and the mechanism allows up to 117. For such data lengths the format is no obstacle: the unrestricted algorithm runs with two queries per query and at most two re-randomizations per query on average, and apart from twice as many queries its costs are those of the raw map.

(iii) Fixed format, digest of a given length. Interfaces that accept a digest computed by the caller, encode it with a fixed prefix and then apply the PKCS#1 v1.5 format give a one-sided window of 2^{8h} offsets for an h-byte digest, with a fixed part determined by the prefix. A 64-byte digest (for example SHA-512) at 1024 bits gives $W = 2^{512}$, exactly half the bits of the block: this is below the bound of Theorem 4.1 by about 2 bits and below that of Corollary 4.2 by about 1 bit, but the measurements of Figure 4 show that the re-randomization succeeds on 79.6% of the draws for random 1024-bit moduli, so the unrestricted algorithm runs with two queries per query and 1.26 draws per query on average. A 32-byte digest (for example SHA-256) gives $W = 2^{256} \approx N^{1/4}$ at 1024 bits, where the ratio method needs at least about 2^{510} draws (Proposition 4.3); this is the AMR setting of Section 3, in which Theorem 3.1 lowers the heuristic cost per root from $L[1.8171]$ to $L[1.5803]$. That result is asymptotic, and we did not run it at this size.

(iv) Digest computed inside. Mechanisms that hash the message inside the device, such as `CKM_SHA256_RSA_PKCS`, leave the caller no choice of the block: the offsets are outputs of the hash function, not inputs chosen by the caller, so these mechanisms are outside both oracle models.

Scope. These statements are about the cost of computing roots after temporary access to the map; they do not recover the key. Three conditions limit their practical reach. First, the largest size at which the unrestricted algorithm has been run is 1024 bits, which current guidance no

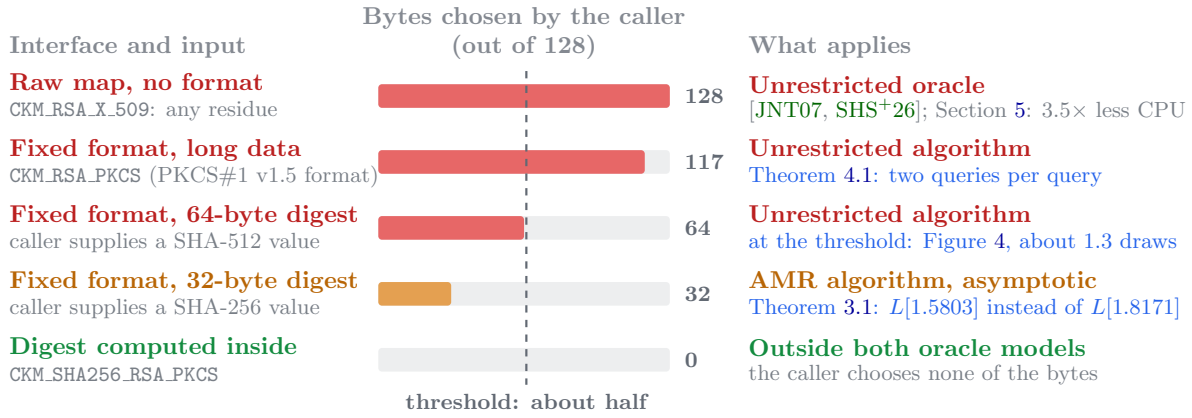


Figure 5: PKCS#11 interfaces for a 1024-bit modulus, ordered by the number of the 128 bytes of the block that the caller chooses, and the result of this paper that applies to each. The dashed line marks the threshold of Section 4.1, about half of the block.

longer approves for generating signatures [BR19]; at 2048 bits every cost estimate of this paper is asymptotic. Second, the model requires direct calls to the signing interface with inputs of the caller's choice, rather than calls through an application that hashes the message before signing. Third, the number of calls is large: about 4×10^9 in the computation of Section 5, all made before the target is known. Of the interfaces above, the raw map is not an approved signature scheme [Nat23]. The fixed format of (ii) is approved for signatures with a digest as input, and Theorem 4.1 shows that the format itself does not restrict a caller who may supply slightly more than half of the block (65 of the 128 bytes at 1024 bits); in our measurements the same holds at exactly half (Figure 4).

5 A 1024-bit computation

This section reports a complete computation of an e-th root, $e = 65537$, with the unrestricted algorithm on the 1024-bit instance of Shea, Haller, Suhl, Heninger and Thomé [SHS+26], with the same algorithm, polynomials and factor-base bounds, and compares the two computations. The comparison is therefore one of implementation and execution, not of algorithms.[2] All queries were made and the oracle was withdrawn before the target was drawn; the online phase made no query, and the output x was checked against $x^e \equiv t \pmod N$.

5.1 Implementation

We implemented the unrestricted algorithm of [JNT07], in the two-polynomial form of [SHS+26], in C++ on top of CADO-NFS [CAD]. It uses a linear polynomial $g = g_1 x + g_0$ and a polynomial f of degree 6 with a common root modulo N. We write B_{rat} for the bound on the rational primes whose roots are queried, B_{alg} for the bound on the algebraic prime ideals of the *first tier*, which enter the linear system, and B_{ext} for the bound on the prime ideals of the *second tier*, each of which is covered by one relation of its own. The second tier does not lower the asymptotic online exponent (see the full version); at concrete sizes it trades offline sieving for a cheaper descent, which is why [SHS+26]

[2] Our oracle program accepted only integers of absolute value at most $N^{1/4}$. This restriction costs nothing: the values that the number field sieve hands to the oracle can be taken to be the rational norms $\text{Res}(g, a - bx) = g_1 a + g_0 b$ of the sieved pairs, which are small integers, together with the primes up to B_{rat} and g_1; the residues $a - bm \equiv (g_1 a + g_0 b)/g_1$ of the formulation with the common root m then follow from the root of g_1. The largest query had 182 bits.

and we use it. The offline phase collects one-sided relations on f by lattice sieving over special-$\mathfrak{q}$, assembles the first tier into a sparse system and the second tier into one record per prime ideal, and has a separate oracle program answer the queries on g_1, on every prime $p \le B_{\text{rat}}$ and on every value $z = g_1 a + g_0 b$ of a used relation. The online phase writes the masked target as a ratio of two smooth integers (the *initial split*), descends every large prime and prime ideal by special-$\mathfrak{q}$ sieving on (g, f), solves the first-tier system for the resulting right-hand side by block Wiedemann [Cop94, Tho02], and extracts the root. The full version lists the stages and the parameters.

Our implementation differs from the code of [SHS+26] in four ways, all of which affect only speed: (i) a faster AI-optimized version of the CADO-NFS lattice siever, which finds the same relations for each special-$\mathfrak{q}$, or, when it stops at the first relation, relations for the same prime ideals; (ii) in the bottom of the descent, each worker stops at the first valid relation, and up to about 8,000 auxiliary primes are tried instead of a fixed 200; (iii) 64 e-th power characters instead of 6 Schirokauer maps, and block Wiedemann on GPUs instead of CPUs; and (iv) the root is computed modulo 64 inert prime powers and combined by an explicit Chinese remainder computation modulo N, instead of Montgomery's product method [Mon94] followed by a hybrid Chinese-remainder and p-adic computation.

5.2 The computation

We used the modulus of [SHS+26], their polynomials (f of degree 6; $|g_1| \approx 2^{69.0}$, $|g_0| \approx 2^{158.7}$) and their bounds $B_{\text{rat}} = 2^{36}$, $B_{\text{alg}} = 2^{31}$ and $B_{\text{ext}} = 2^{35}$. CPU time is user plus system time of all jobs whose output was used, in CPU core-years on virtual cores; GPU time is listed separately. The target was derived after the withdrawal from a public seed string by SHA-256 in counter mode, reduced modulo N, and recorded before the online phase started. Table 2 summarizes the costs and Figure 6 the calendar.

Offline phase. The first tier consists of the 105,113,323 algebraic prime ideals of degree one and norm at most B_{alg}. Lattice sieving over special-$\mathfrak{q}$ in $[4 \cdot 10^8, 2^{31})$, followed by two coverage rounds in which every prime ideal that occurred in no relation was sieved as its own special-$\mathfrak{q}$, gave about 163 million relations. The filtering programs of CADO-NFS reduced them to a matrix with 9.74 million rows, and 99.43% of the first-tier prime ideals were linked to the system, either as columns or through relations that eliminate them during back-substitution. Two linking rounds with composite special-$\mathfrak{q}$ reduced the 602,612 unlinked prime ideals to 89 and then to 64, which the online descent avoids. Filtering the 167,962,900 relations gave a matrix of 9,634,465 rows. The second tier consists of the prime ideals of degree one and norm in $(2^{31}, 2^{35}]$, about as many as the 1,375,108,714 primes in this interval. For each of them, the siever stopped at the first relation in which it is the largest prime ideal; this gave 1,075,411,377 records, 78.2% of the number of these primes (from 92.5% on $(2^{31}, 2^{32}]$ to 72.7% on $(2^{34}, 2^{35}]$), against 76.2% in [SHS+26].

Oracle. The oracle answered 4,055,916,065 queries: g_1, the 2,874,398,515 primes up to 2^{36}, 106,106,172 values z of first-tier relations and 1,075,411,377 values z of second-tier records. It was a separate program, the only one with access to the factorization of N. The queries ran as 63 parallel tasks in 20 minutes.

Online phase. The initial split, with large primes up to 2^{135} as in [SHS+26], ran as 2,000 parallel tasks, the first success cancelling the others; it succeeded after 46 minutes, with 33 rational primes, the largest of 119 bits. A second split of the same target, searched in parallel with large primes up to 2^{106}, had all its primes below 2^{90} and was used instead, because special-$\mathfrak{q}$ above about 2^{95}

Table 2: The 1024-bit computation, in CPU core-years (measured CPU usage of all jobs whose output was used), with the costs reported by [SHS+26] for the same modulus (their Figure 4, without polynomial selection, which we reused). The totals of [SHS+26] are their rounded figures; their stage costs add up to 1,204 offline and 1,383 in all. Our linear algebra ran on GPUs (about 330 GPU-hours reserved); trial runs and probes, 7.5 core-years, are not included.

Stage	[SHS+26]	this work	ratio
core sieving, coverage and linking rounds	150	55	2.7
second-tier (extension) sieving	1,054	262	4.0
offline total	1,200	318	3.8
descent, including the initial split	162	73	2.2
linear algebra	17	GPUs	—
root extraction	0.25	0.04	≈ 6
online total, first root	180	73	2.5
first root, total	1,380	391	3.5
queries	4.07×10^9	4.06×10^9	

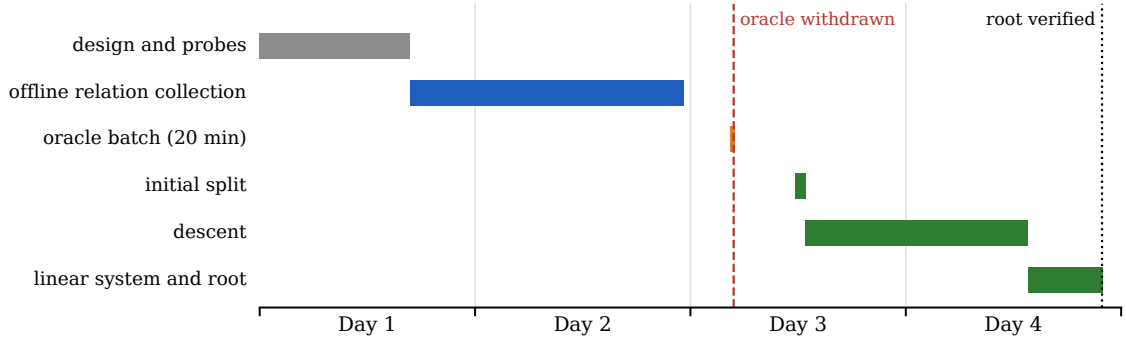


Figure 6: Timeline of the 1024-bit computation. Days are counted in periods of 24 hours from the start of the design; the root was verified 3 days and 22 hours after it. The oracle batch took 20 minutes; every later stage ran without oracle access.

are impractical for the lattice siever [SHS+26]. The descent then ran in rolling waves: every newly created node above the bottom level got its own sieving task, and stubborn nodes were handled by groups of tasks with a skewed special-$\mathfrak{q}$ lattice. The middle descent finished 21 hours and the bottom descent 25 hours after the first split was found; in total the descent used 1,832 relations and left 19,334 second-tier prime ideals, which were replaced by their records. The linear system was solved by block Wiedemann with blocking factors $m = n = 16$ on 16 GPUs: the Krylov sequence of 1,205,248 steps took about 4.1 hours, followed by the linear generator and the solution, and the e-th root was computed modulo 64 prime powers in parallel, in about 17 minutes per task, and combined. The root was verified 34.5 hours after the target was drawn; we re-checked $x^e \equiv t \pmod N$ independently from the public data.

5.3 Comparison with Shea, Haller, Suhl, Heninger and Thomé

Where the factor of 3.5 comes from. On the same modulus, polynomials and factor-base bounds, the first root cost 391 CPU core-years against 1,380, a factor of 3.5 (Table 2 and Figure 7). The two computations run the same algorithm, so this factor measures implementation efficiency;

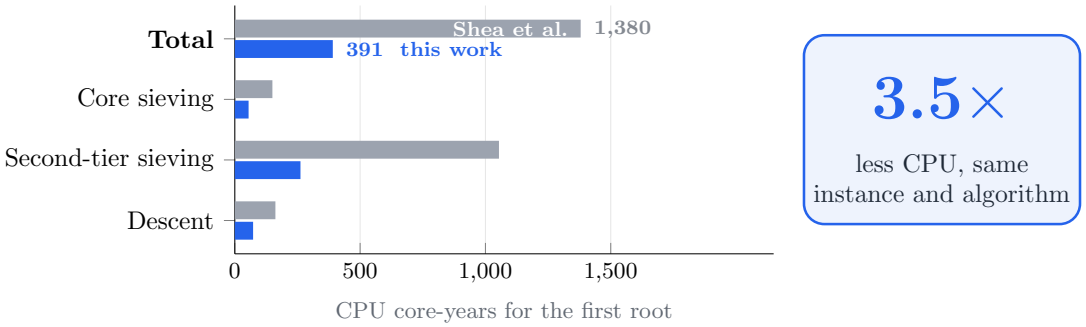


Figure 7: CPU core-years for the first root on the 1024-bit instance of [SHS+26], including the one-time offline phase: their reported costs and ours; the totals also include linear algebra and root extraction, which are not shown separately. Each further root then cost 73 against 180 core-years. Our linear algebra ran on GPUs and is not counted; charging it at the 17 core-years of [SHS+26] still gives a factor of 3.4.

CPU time counts the work done, however many cores perform it. The stage ratios of Table 2 do not multiply: each applies only to its own share of the cost, so the total ratio is their harmonic mean weighted by the costs of [SHS+26], and it is dominated by the second-tier sieving, which is 76% of their cost. Sieving accounts for 90% of the saving (80% from the second tier alone). We compared our siever, an optimized fork of the CADO-NFS lattice siever, with the upstream siever on one machine, on the same special-$\mathfrak{q}$ and with the same parameters, with one and with 32 threads per process. On the core sieve, where each special-$\mathfrak{q}$ is sieved in full, both find exactly the same relations, and ours is 1.8 times faster with one thread and 1.9 times faster with 32. In the second-tier mode, both followed the strategy of stopping as soon as a relation is encountered, and found relations for exactly the same prime ideals. Our implementation of that "early-exit" strategy propagates better across threads since it is 3.4 times faster with one thread and 4.4 times faster with 32. These same-machine factors bound the role of hardware. Taking the smaller, one-thread factors, the siever explains 1.8 of the measured 2.7 on the core sieve and 3.4 of the measured 4.0 on the second tier, so at most a factor of 1.5 on the core sieve and 1.2 on the second tier can come from hardware or other differences. (Running several special-$\mathfrak{q}$ per process instead of one lowered our own second-tier cost by a factor of about 1.5; this compares two of our configurations, not ours with [SHS+26].) The remaining gains come from choices within the same method, for which we have no same-machine comparison: the bottom descent stops at the first valid relation, whereas the code of [SHS+26] tries all 200 auxiliary primes of a job (stage ratio 2.2, which also reflects our siever and the hardware), and the root is computed modulo 64 inert prime powers and combined by an explicit Chinese remainder computation modulo N (stage ratio about 6). Our linear algebra ran on GPUs (about 330 GPU-hours reserved, about 100 busy); charging it at the 17 core-years of [SHS+26] gives a factor of 3.4, and adding our 7.5 core-years of probes and rehearsals gives 3.3. None of these gains changes the asymptotic cost. The core-siever optimizations carry over to any CADO-NFS computation.

Calendar time. The computation took 3.2 days from the first full-scale sieving job to the verified root, against about five months for [SHS+26]. This difference, unlike the factor of 3.5, reflects mainly parallelism: we kept on average about 45,000 cores busy (3.4 million core-hours in 3.2 days, with peaks of about 300,000 CPU cores and 16 GPUs), against an average of about 3,300 cores for [SHS+26] (12.2 million core-hours in five months). We do not count it as an efficiency gain; it

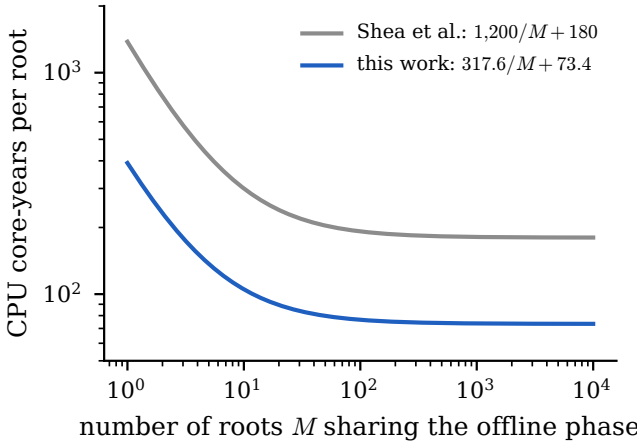


Figure 8: CPU core-years per root on the 1024-bit instance when M roots share one offline phase: $P/M + R$ with the offline and online costs of Table 2 (ours unrounded: $P = 317.6$, $R = 73.4$). Our linear algebra ran on GPUs and is not counted; charged at the 17 core-years of [SHS+26], our limit would be 90 against 180 core-years.

shows that the agents of Section 6 could run and debug the computation at that scale.

Many roots, and factoring. When M roots share one offline phase, the cost per root is $P/M+R$; ours is lower than that of [SHS+26] for every M and tends to 73 against 180 core-years (Figure 8). The cost of factoring a 1024-bit modulus is estimated at 500,000 to 1,000,000 core-years (as cited in [SHS+26]); our first root cost 0.04% to 0.08% of this, and each further root 0.007% to 0.015%. Computing M roots is therefore cheaper than factoring the modulus as long as M is below about 6,800 to 13,600, against about 2,800 to 5,500 with the costs of [SHS+26]; these numbers are more than five orders of magnitude below the number of queries, in line with a remark in the full version, which shows that, asymptotically, the break-even number of roots lies below the number of queries.

Scope. The computation is a complete run of the unrestricted algorithm. It does not test Theorem 3.1, which concerns short windows; at cryptographic sizes, the fields of degree at least two that a generic c requires make the offline phase far more expensive, and we leave such computations to future work. Our CPU times depend on the processors and on the implementation, and we compare with costs reported on other hardware; the same-machine siever factors above bound that dependence for the sieving, which accounts for 90% of the saving.

6 The agentic setup

The 1024-bit computation of Section 5.2 was designed, implemented, run and debugged by AI agents. This section describes the setup and quantifies the work; we report it because the scale of the engineering, not only the arithmetic, determined whether the computation could be done.

Roles. Two language-model agents worked as a supervisor and an executor (Figure 9).

- *Model A, the supervisor,* wrote and maintained the design, including the parameters, the resource budget of every stage and the checks that each stage had to pass before the next

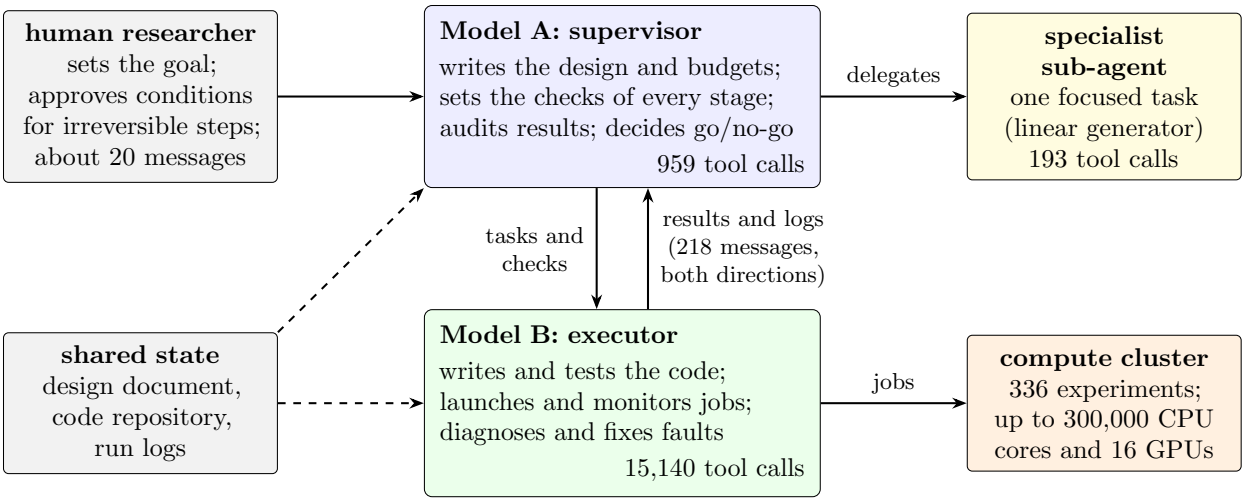


Figure 9: The agent system of the 1024-bit computation. Counts are from the agent transcripts, from the start of the design to the verified root.

one could spend compute. It reviewed every result reported by the executor, decided the go/no-go points, and started a specialist sub-agent for one focused task, a 5× speed-up of the linear-generator step of block Wiedemann for the prime $e = 65537$.

- *Model B, the executor,* wrote the code, built and tested it, launched and monitored the distributed jobs, audited their outputs, diagnosed failures and reported to the supervisor.
- *The human researcher* set the goal and approved the gates, mostly by replying to the supervisor's recommendation.

The agents communicated through messages and through shared state: the design document, the code repository and the logs of the runs. Both agents operated through tool calls: shell commands, file edits, code searches and job launches on a shared compute cluster.

Gated workflow. Each stage was first exercised on a small probe or a 1% canary slice, and scaled up only after it passed checks that the supervisor had fixed in advance: for example, that the fork and the upstream siever find the same relations on the same special-$\mathfrak{q}$, or relations for the same prime ideals when they stop at the first relation; per-stage CPU and memory budgets; and, before the oracle batch, that every relation referenced by the linear system was among the queried ones. The oracle batch, which cannot be repeated, was gated on all of these checks, and the 1024-bit target was drawn only after a complete rehearsal of the online code at 498 bits had computed roots for several targets. Figure 10 shows the four days.

Scale. Table 3 summarizes the work, counted from the start of the design, at the beginning of Day 1 in Figures 10 and 6, to the verified root on Day 4, 3 days and 22 hours later. The agents wrote 33,500 lines of code for the 1024-bit run in 19 code changes (including 24,000 lines of C++ and CUDA and 8,400 lines of Python), on top of an existing implementation of the pipeline for smaller moduli, for a total of 66,400 lines in 48 changes. They launched and monitored 336 distributed experiments, at peak on about 300,000 CPU cores and 16 GPUs, made 16,292 tool calls and exchanged 218 messages. The human researcher sent about 20 messages, mostly approvals such as "follow your recommendation" and "continue". With this little human input, the agents brought

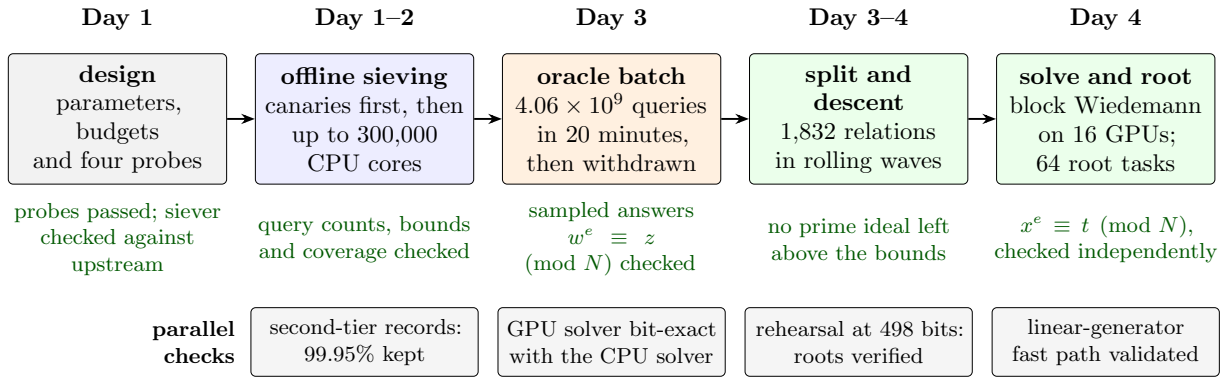


Figure 10: The gated workflow of the 1024-bit computation. Each stage had to pass its check before the next stage spent compute; the checks in the bottom row ran in parallel with the main line.

Table 3: The work of the agents on the 1024-bit computation, from the start of the design to the verified root.

Elapsed time	3 days 22 hours
New code for the 1024-bit run	33,500 lines in 19 changes
of which C++ and CUDA / Python	24,000 / 8,400 lines
Whole pipeline, including the code for smaller moduli	66,400 lines in 48 changes
Distributed experiments	336
Peak resources	about 300,000 CPU cores and 16 GPUs
Tool calls: executor / supervisor / specialist	15,140 / 959 / 193
Messages between agents	218
Messages from the human researcher	about 20

the CPU time of the first root to 3.5 times below that reported by [SHS+26] on the same instance. Using the public code of [SHS+26] as the reference for the parameters, the auxiliary primes and the coverage rounds, they stopped the descent at the first valid relation instead of trying all 200 auxiliary primes, computed the root modulo prime powers, and moved the linear algebra to GPUs; most of the saving, however, comes from faster sieving (Section 5.3). The total factor of 3.5 compares our measured CPU time with the costs reported in [SHS+26], which were obtained on their hardware. To separate the effect of the code from that of the hardware, our siever was also compared on one machine with the upstream CADO-NFS siever, on which [SHS+26] built: on the same special-$\mathfrak{q}$ both find exactly the same relations, or, when they stop at the first relation, relations for exactly the same prime ideals, and ours is faster by the factors given in Section 5.3. Running at a scale of up to 300,000 cores, they also completed the computation in 3.2 days of calendar time from the first full-scale sieving job, against about five months for [SHS+26].

Problems found at full scale. The online code had passed a complete rehearsal at 498 bits, in which roots were computed for several targets, before the 1024-bit target was drawn. Some problems nevertheless appeared only at 1024 bits, because the rehearsal could not reach the sizes involved or used a polynomial of smaller degree. Table 4 lists those that bear on the algorithm or on the correctness of the result: two gaps in the descent strategy (rows 1 and 2), a final combination step whose cost had never been measured at full size (row 3), and a 32-bit index overflow that made the first computed root wrong (row 4). The agents diagnosed and resolved each of them while the computation continued; none required repeating the oracle batch.

Table 4: Problems that appeared only in the 1024-bit online phase, why the 498-bit rehearsal did not expose them, and how they were resolved.

	Problem	How it appeared	Why only at 1024 bits	Resolution
1	Abandoned descent nodes	The descent mode of the siever gave up on three rational nodes of 82 to 85 bits.	Nodes of this size do not occur at 498 bits, where the initial split has primes of at most about 61 bits.	A large-q step on a skewed special-$\mathfrak{q}$ lattice, run as parallel groups of tasks, solved all three nodes and an 81-bit algebraic node.
2	Bottom-descent parameters	Composite special-$\mathfrak{q}$ with the auxiliary prime on the rational side made the degree-6 algebraic norm grow by about 3 bits per bit of the auxiliary prime; some bottom ideals stalled.	The parameters had been calibrated on a smaller instance with a degree-5 polynomial.	Auxiliary primes tried from the smallest up, larger sieve areas or the auxiliary prime on the algebraic side, and an algebraic large-prime bound of up to 37 bits in the bottom descent and up to 40 bits for the last middle-descent nodes.
3	Slow final combination	Rebuilding each coefficient of the root from 64 residues of about 37 million bits on one thread projected to more than an hour.	Residues of this size occur only at 1024 bits; the step had never been timed at full size.	An explicit Chinese remainder computation modulo N, in parallel over the 64 primes: 116 seconds.
4	Wrong root from a 32-bit overflow	The first complete root failed the check $x^e \equiv t \pmod N$. A 32-bit byte offset into the stored answers wrapped at index 2^{25}, so the last 5.96 million of the 39.5 million relations whose stored answers enter the root read the wrong answers.	Offsets beyond 4 GiB require more than 2^{25} such relations; the rehearsal had far fewer.	A 64-bit offset. The fault was localized by checking an identity of the saved descent state modulo N and then its rational and algebraic halves separately.

What the human did and did not do. The researcher chose the problem, the oracle bound and the instance, approved the conditions for the irreversible steps (the oracle batch and the drawing of the target), leaving the final go-ahead to the supervisor, and asked questions. The researcher wrote no code, launched no job and diagnosed no fault.

7 Discussion

This paper asks how much a fixed format restricts an oracle for e-th roots modulo N. The answer depends less on the presence of a format than on how many bits of each input the caller chooses. Once the caller chooses about half of the bits of each input, the affine oracle is as strong as the unrestricted one, at two queries per query (Theorem 4.1), and the computation of Section 5 carries over with twice as many queries. This is a property of the format itself, so it holds for every correct implementation of it. Below about half, the reduction degrades quickly (Proposition 4.3); there the only algorithm we know is the AMR algorithm of Section 3, which costs more than the unrestricted algorithm for a single target, even asymptotically.

Scope. These results do not change the security of modern RSA deployments. Signers that hash inside the device are outside both oracle models. The largest size at which the unrestricted

algorithm has been run is 1024 bits, a size no longer approved for generating signatures [BR19]; at 2048 bits every cost estimate in this paper is asymptotic. The model also assumes direct calls with inputs of the caller's choice, all made before the target is known; the 1024-bit computation made about 4×10^9 of them (Section 5). The evidence is of three kinds: the threshold results of Section 4.1 are unconditional; the case of a variable part of exactly half the bits of N at 1024 bits rests on measurement (Figure 4), not on proof; and the costs of the AMR algorithm rest on the heuristics H1–H5, like the standard analyses of the number field sieve.

Open questions. We single out two.

1. *Windows below half, in practice.* For windows of N^θ offsets with $\theta < 1/2$, the AMR algorithm has not been run at any cryptographic size, and this regime is empirically open. Its offline phase collects relations in number fields of degree at least two, which is far more expensive than for the unrestricted algorithm; making it practical is the main open problem on the computational side.
2. *Between the two regimes.* For $0 < \theta < 1/2$, is there an algorithm whose cost interpolates between that of the AMR algorithm and that of the unrestricted one, rather than switching sharply at $\theta = 1/2$?

Outlook. More broadly, the threshold of Theorem 4.1 is a statement about a standard, not about a program: it measures how much freedom a format leaves to the caller. This suggests analysing the formats in standards directly, in addition to checking that implementations follow them. The computation of Section 5 was designed, run and debugged by AI agents (Section 6). As such systems improve, we expect them to help measure the strength of deployed cryptographic choices more precisely, and so to plan transitions to stronger ones well ahead of time.

Acknowledgements

We thank Ahmed Eleryan, Akvilė Žemgulytė, Andrew Cowie, Charlie Beattie, Craig Donner, James Spencer, Malo Denielou, Peter Hausel, Sebastian Bodenstein, Tudor Marian and Victor Martin, who were immensely helpful in making our engineering efforts a success.

References

[AH99] Leonard M. Adleman and Ming-Deh A. Huang. Function field sieve method for discrete logarithms over finite fields. *Information and Computation*, 151(1–2):5–16, 1999.

[Bar13] Răzvan Barbulescu. *Algorithmes de logarithmes discrets dans les corps finis*. PhD thesis, Université de Lorraine, 2013. https://theses.hal.science/tel-00925228.

[BL14] Daniel J. Bernstein and Tanja Lange. Batch NFS. In *Selected Areas in Cryptography – SAC 2014*, volume 8781 of *Lecture Notes in Computer Science*, pages 38–58. Springer, 2014.

[BLP93] Joe P. Buhler, Hendrik W. Lenstra, Jr., and Carl Pomerance. Factoring integers with the number field sieve. In *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*, pages 50–94. Springer, 1993.

[BR19] Elaine Barker and Allen Roginsky. Transitioning the use of cryptographic algorithms and key lengths. NIST Special Publication 800-131A Rev. 2, National Institute of Standards and Technology, March 2019.

[CAD] The CADO-NFS Development Team. *CADO-NFS, an implementation of the number field sieve algorithm*. https://cado-nfs.gitlabpages.inria.fr/.

[CEP83] E. Rodney Canfield, Paul Erdős, and Carl Pomerance. On a problem of Oppenheim concerning "factorisatio numerorum". *Journal of Number Theory*, 17(1):1–28, 1983.

[Cop84] Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–594, 1984.

[Cop94] Don Coppersmith. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.

[CS06] An Commeine and Igor Semaev. An algorithm to solve the discrete logarithm problem with the number field sieve. In *Public Key Cryptography – PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 174–190. Springer, 2006.

[Gor93] Daniel M. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM Journal on Discrete Mathematics*, 6(1):124–138, 1993.

[JL03] Antoine Joux and Reynald Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method. *Mathematics of Computation*, 72(242):953–967, 2003.

[JNT07] Antoine Joux, David Naccache, and Emmanuel Thomé. When e-th roots become easier than factoring. In *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2007.

[Len87] Hendrik W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126(3):649–673, 1987.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LLMP90] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., Mark S. Manasse, and John M. Pollard. The number field sieve. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC 1990)*, pages 564–572, 1990.

[MKJR16] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA cryptography specifications version 2.2. RFC 8017, Internet Engineering Task Force, November 2016.

[Mon94] Peter L. Montgomery. Square roots of products of algebraic numbers. In *Mathematics of Computation 1943–1993: A Half-Century of Computational Mathematics*, volume 48 of *Proceedings of Symposia in Applied Mathematics*, pages 567–571. American Mathematical Society, 1994.

[Nat23] National Institute of Standards and Technology. Digital signature standard (DSS). Federal Information Processing Standards Publication 186-5, National Institute of Standards and Technology, February 2023.

[OAS15] OASIS. *PKCS #11 Cryptographic Token Interface Current Mechanisms Specification Version 2.40*, April 2015. Edited by Susan Gleeson and Chris Zimman. OASIS Standard.

[Pom82] Carl Pomerance. Analysis and comparison of some integer factoring algorithms. In *Computational Methods in Number Theory, Part I*, volume 154 of *Mathematical Centre Tracts*, pages 89–139. Mathematisch Centrum, Amsterdam, 1982.

[Sch93] Oliver Schirokauer. Discrete logarithms and local units. *Philosophical Transactions of the Royal Society of London, Series A*, 345(1676):409–423, 1993.

[Sem98] Igor A. Semaev. An algorithm for evaluation of discrete logarithms in some nonprime finite fields. *Mathematics of Computation*, 67(224):1679–1689, 1998.

[SHS+26] Laura Shea, Miro Haller, Adam Suhl, Nadia Heninger, and Emmanuel Thomé. Forging 1024-bit RSA signatures in nearly SNFS time. Cryptology ePrint Archive, Paper 2026/2131, 2026.

[Tho02] Emmanuel Thomé. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *Journal of Symbolic Computation*, 33(5):757–775, 2002.