Batch NFS

Daniel J. Bernstein[1,2] and Tanja Lange[2]

[1] Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607–7045, USA
djb@cr.yp.to

[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
tanja@hyperelliptic.org

Abstract. This paper shows, assuming standard heuristics regarding the number-field sieve, that a "batch NFS" circuit of area $L^{1.181\ldots+o(1)}$ factors $L^{0.5+o(1)}$ separate B-bit RSA keys in time $L^{1.022\ldots+o(1)}$. Here $L = \exp((\log 2^B)^{1/3}(\log\log 2^B)^{2/3})$. The circuit's area-time product (price-performance ratio) is just $L^{1.704\ldots+o(1)}$ per key. For comparison, the best area-time product known for a single key is $L^{1.976\ldots+o(1)}$.

This paper also introduces new "early-abort" heuristics implying that "early-abort ECM" improves the performance of batch NFS by a superpolynomial factor, specifically $\exp((c + o(1))(\log 2^B)^{1/6}(\log\log 2^B)^{5/6})$ where c is a positive constant.

Keywords: integer factorization, number-field sieve, price-performance ratio, batching, smooth numbers, elliptic curves, early aborts

1 Introduction

The cryptographic community reached consensus a decade ago that a 1024-bit RSA key can be broken in a year by an attack machine costing significantly less than 10^9 dollars. See [51], [38], [24], and [23]. The attack machine is an optimized version of the number-field sieve (NFS), a factorization algorithm that has been intensively studied for twenty years, starting in [36]. The run-time analysis of NFS relies on various heuristics, but these heuristics have been confirmed in a broad range of factorization experiments using several independent NFS software implementations: see, e.g., [29], [30], [31], and [4].

Despite this threat, 1024-bit RSA remains the workhorse of the Internet's "DNS Security Extensions" (DNSSEC). For example, at the time of this writing (November 2014), the IP address of the domain dnssec-deployment.org is

- signed by that domain's 1024-bit "zone-signing key", which in turn is

This work was supported by the National Science Foundation under grant 1018836 and by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005. Permanent ID of this document: 4f99b1b911984e501c099f514d8fd2ce. Date: 2014.11.09.

- signed by that domain's 2048-bit "key-signing key", which in turn is
- signed by `.org`'s 1024-bit zone-signing key, which in turn is
- signed by `.org`'s 2048-bit key-signing key, which in turn is
- signed by the DNS root's 1024-bit zone-signing key, which in turn is
- signed by the DNS root's 2048-bit key-signing key.

An attacker can forge this IP address by factoring any of the three 1024-bit RSA keys in this chain.

A report [41] last year indicated that, out of the 112 top-level domains using DNSSEC, 106 used the same key sizes as `.org`. We performed our own survey of zone-signing keys in September 2014, after many new top-level domains were added. We found 286 domains using 1024-bit keys; 4 domains using 1152-bit keys; 192 domains using 1280-bit keys; and just 22 domains using larger keys. Almost all of the 1280-bit keys are for obscure domains such as `.boutique` and `.rocks`; high-volume domains practically always use 1024-bit keys.

Evidently DNSSEC users find the attacks against 1024-bit RSA less worrisome than the obvious costs of moving to larger keys. There are, according to our informal surveys of these users, three widespread beliefs supporting the use of 1024-bit RSA:

- A typical RSA key is believed to be worth less than the cost of the attack machine.
- Building the attack machine means building a huge farm of application-specific integrated circuits (ASICs). Standard computer clusters costing the same amount of money are believed to take much longer to perform the same calculations.
- It is believed that switching RSA signature keys after (e.g.) a month will render the attack machine useless, since the attack machine requires a full year to run.

Consider, for example, the following quote from the latest "DNSSEC operational practices" recommendations [32, Section 3.4.2], published December 2012:

> DNSSEC signing keys should be large enough to avoid all known cryptographic attacks during the effectivity period of the key. To date, despite huge efforts, no one has broken a regular 1024-bit key; in fact, the best completed attack is estimated to be the equivalent of a 700-bit key. An attacker breaking a 1024-bit signing key would need to expend phenomenal amounts of networked computing power in a way that would not be detected in order to break a single key. Because of this, it is estimated that most zones can safely use 1024-bit keys for at least the next ten years.

This quote illustrates the first and third beliefs reported above: the attack cost would be "phenomenal" and would break only "a single key"; furthermore, the attack would have to be completed "during the effectivity period of the key". A typical DNSSEC key is valid for just one month and is then replaced by a new key.

1.1. Contents of this paper. This paper analyzes the *asymptotic* cost, specifically the *price-performance ratio*, of breaking *many* RSA keys. We emphasize several words here:

- "Many": The attacker is faced not with a single target, but with many targets. The algorithmic task here is not merely to break, e.g., a single 1024-bit RSA key; it is to break more than two hundred 1024-bit RSA keys for DNSSEC top-level domains, many more 1024-bit RSA keys at lower levels of DNSSEC, millions of 1024-bit RSA keys in SSL (as in [25] and [35]; note that upgrading SSL to 2048-bit RSA does nothing to protect the confidentiality of previously recorded SSL traffic), etc. This is important if there are ways to share attack work across the keys.
- "Price-performance ratio": As in [53], [18], [50], [15], [54], [7], [51], [56], [23], [24], etc., our main interest is not in the number of "operations" carried out by an algorithm, but in the actual price and performance of a machine carrying out those operations. Parallelism increases price but often improves performance; large storage arrays are a problem for both price and performance. We use price-performance ratio as our primary cost metric, but we also report time separately since signature-key rotation puts a limit upon time.
- "Asymptotic": The cost improvements that we present are superpolynomial in the size of the numbers being factored. We thus systematically suppress all polynomial factors in our cost analyses, simplifying the analyses.

This paper presents a new "batch NFS" circuit of area $L^{1.181\ldots+o(1)}$ that, assuming standard NFS heuristics, factors $L^{0.5+o(1)}$ separate B-bit RSA keys in total time just $L^{1.022\ldots+o(1)}$. The area-time product is $L^{1.704\ldots+o(1)}$ for each key; i.e., the price-performance ratio is $L^{1.704\ldots+o(1)}$. Here (as usual for NFS) L means $\exp((\log N)^{1/3}(\log\log N)^{2/3})$ where $N = 2^B$.

For comparison (see Table 1.4), the best area-time product known for factoring a single key (without quantum computers) is $L^{1.976\ldots+o(1)}$, even if non-uniform precomputations such as Coppersmith's "factorization factory" are allowed. The literature is reviewed below.

This paper also looks more closely at the $L^{o(1)}$. The main bottleneck in batch NFS is not traditional sieving, but rather low-memory smoothness detection, motivating new attention to the complexity of low-memory smoothness detection. Traditional ECM, the elliptic-curve method of recognizing y-smooth integers, works in low memory and takes time $\exp(\sqrt{(2+o(1))\log y \log\log y})$. One can reasonably guess that, compared to traditional ECM, "early-abort ECM" saves a subexponential factor here, but the complexity of early-abort ECM has never been analyzed. Section 3 of this paper introduces new early-abort heuristics implying that the cost of early-abort ECM is $\exp\left(\sqrt{\left(\frac{8}{9}+o(1)\right)\log y \log\log y}\right)$. Using early aborts increases somewhat the number of auxiliary integers that need to be factored, producing a further increase in cost, but the cost is outweighed by the faster factorization.

The ECM cost is obviously bounded by $L^{o(1)}$: more precisely, the cost is $\exp(\Theta((\log N)^{1/6}(\log\log N)^{5/6}))$ in the context of batch NFS, since $y \in L^{\Theta(1)}$.

This cost is invisible at the level of detail of $L^{1.704\ldots+o(1)}$. The speedup from ECM to early-abort ECM is nevertheless superpolynomial and directly translates into the same speedup in batch NFS.

1.2. Security consequences. We again emphasize that our results are asymptotic. This prevents us from directly drawing any conclusions about 1024-bit RSA, or 2048-bit RSA, or any other specific RSA key size. Our results are nevertheless sufficient to undermine all three of the beliefs described above:

- Users comparing the value of an RSA key to the cost of an attack machine need to know the *per-key* cost of batch NFS. This has not been seriously studied. What the literature has actually studied in detail is the cost of NFS attacking *one key at a time*; this is not the same question. Our asymptotic results do not rule out the possibility that these costs are the same for 1024-bit RSA, but there is also no reason to be confident about any such possibility.
- Most of the literature on single-key NFS relies heavily on operations that — for large key sizes — are not handled efficiently by current CPUs and that become much more efficient on ASICs: consider, for example, the routing circuit in [51]. Batch NFS relies much more heavily on massively parallel elliptic-curve scalar multiplication, exactly the operation that is shown in [12], [11], and [17] to fit very well into off-the-shelf graphics cards. The literature supports the view that off-the-shelf hardware is much less cost-effective than ASICs for single-key NFS, but there is no reason to think that the same is true for batch NFS.
- The natural machine size for batch NFS (i.e., the circuit area if price-performance ratio is optimized) is larger than the natural machine size for single-key NFS, but the natural *time* is considerably smaller. As above, these asymptotic results undermine any confidence that one can obtain from comparing the natural time for single-key NFS to the rotation interval for signature keys: there is no reason to think that the latency of batch NFS will be as large as the latency of single-key NFS. Note that, even though this paper emphasizes optimal price-performance ratio for simplicity, there are also techniques to further reduce the time below the natural time, hitting much lower latency targets without severely compromising price-performance ratio: in particular, for the core sorting subroutines inside linear algebra, one can replace time T with T/f at the expense of replacing area A with Af^2.

The standard measure of security is the total cost of attacking *one* key. For example, this is what NIST is measuring in [6] when it reports “80-bit security” for 1024-bit RSA, “112-bit security” for 2048-bit RSA, “128-bit security” for 3072-bit RSA, etc. What batch NFS illustrates is that, when there are many user keys, the attacker’s cost *per key* can be smaller than the attacker’s total cost for one key. It is much more informative to measure the attacker’s total cost of attacking U user keys, as a function of U. It is even more informative to measure the attacker’s chance of breaking exactly K out of U simultaneously attacked keys in time T using a machine of cost A, as a function of (K, U, T, A).

There are many other examples of cryptosystems where the attack cost does not grow linearly with the number of targets. For example, it is well known that exhaustive search finds preimages for U hash outputs in about the same time as a preimage for a single hash output; furthermore, the first preimage that it finds appears after only $1/U$ of the total time, reducing actual security by $\lg U$ bits. However, most cryptosystems have moved up to at least a "128-bit" security level, giving them a buffer against losing some bits of security. RSA is an exception: its poor performance at high security levels has kept it at a bleeding-edge "80-bit security" level. Even when users can be convinced to move away from 1024-bit keys, they normally move to ≤2048-bit keys. We question whether it is appropriate to view 1024-bit keys as "80-bit" security and 2048-bit keys as "112-bit" security if the attacker's costs *per key* are not so high.

1.3. Previous work. In the NFS literature, as in the algorithm literature in general, there is a split between traditional analyses of "operations" (adding two 64-bit integers is one "operation"; looking up an element of a 2^{64}-byte array is one "operation") and modern analyses of more realistic models of computation. We follow the terminology of our paper [14]: the "RAM metric" counts traditional operations, while the "AT metric" multiplies the area of a circuit by the time taken by the same circuit.

Buhler, H. Lenstra, and Pomerance showed in [19] (assuming standard NFS heuristics, which we now stop mentioning) that NFS factors a single key N with RAM cost $L^{1.922...+o(1)}$. As above, L means $\exp((\log 2^B)^{1/3}(\log\log 2^B)^{2/3})$ if N has B bits. This exponent $1.922\ldots$ is the most frequently quoted cost exponent for NFS.

Coppersmith in [20] introduced two improvements to NFS. The first, "multiple number fields", reduces the exponent $1.922\ldots + o(1)$ to $1.901\ldots + o(1)$. The second, the "factorization factory", is a *non-uniform* algorithm that reduces $1.901\ldots + o(1)$ to just $1.638\ldots + o(1)$. Recall that (size-)non-uniform algorithms are free to perform arbitrary amounts of precomputation as functions of the *size* of the input, i.e., the number of bits of N. A closer look shows that Coppersmith's precomputation costs $L^{2.006...+o(1)}$, so if it is applied to more than $L^{0.368...+o(1)}$ inputs then the precomputation cost can quite reasonably be ignored.

Essentially all of the subsequent NFS literature has consisted of analysis and optimization of algorithms that cost $L^{1.922...+o(1)}$, such as the algorithm of [19]. The ideas of [20] have been dismissed for three important reasons:

- The bottleneck in [19] is sieving, while the bottleneck in [20] is ECM. Both of these algorithms use $L^{o(1)}$ operations in the RAM metric, but the $o(1)$ is considerably smaller for sieving than for ECM.
- Even if the $o(1)$ in [20] were as small as the $o(1)$ in [19], there would not be much benefit in $1.901\ldots + o(1)$ compared to $1.922\ldots + o(1)$. For example, $(2^{50})^{1.922} \approx 2^{96}$ while $(2^{50})^{1.901} \approx 2^{95}$.
- The change from $1.901\ldots + o(1)$ to $1.638\ldots + o(1)$ is much larger, but it comes at the cost of massive memory consumption. Specifically, [20] requires space $L^{1.638...+o(1)}$, while [19] uses space just $L^{0.961...+o(1)}$. This is not visible in the RAM metric but is obviously a huge problem in reality, and it becomes

metric	exponent	precomp	batch	source
AT	$1.976\ldots$	0	0	2001 Bernstein [7]
RAM (unrealistic)	$1.922\ldots$	0	0	1993 Buhler–H. Lenstra–Pomerance [19]
RAM (unrealistic)	$1.901\ldots$	0	0	1993 Coppersmith [20]
AT	$1.900\ldots$	0	0.1	**batch NFS**; this paper
AT	$1.829\ldots$	0	0.2	**batch NFS**; this paper
AT	$1.763\ldots$	0	0.3	**batch NFS**; this paper
AT	$1.710\ldots$	0	0.4	**batch NFS**; this paper
AT	$1.704\ldots$	0	0.5	**batch NFS**; this paper
RAM (unrealistic)	$1.638\ldots$	$2.006\ldots$	0	1993 Coppersmith [20]

Table 1.4. Asymptotic exponents for several variants of NFS, assuming standard heuristics. "Exponent" e means asymptotic cost $L^{e+o(1)}$ per key factored. "Precomp" 2θ means that there is a precomputation involving integer pairs (a, b) up to $L^{\theta+o(1)}$, for total precomputation cost $L^{2\theta+o(1)}$; algorithms without precomputation have $2\theta = 0$. "Batch" β means batch size $L^{\beta+o(1)}$; algorithms handling each key separately have $\beta = 0$. See Section 2 for further details.

> increasingly severe as computations grow larger. As a concrete illustration of the real-world costs of storage and computation, paying for 2^{70} bytes of slow storage (about $30 \cdot 10^9$ USD in hard drives) is much more troublesome than paying for 2^{80} floating-point multiplications (about $0.02 \cdot 10^9$ USD in GPUs plus $0.005 \cdot 10^9$ USD for a year of electricity).

We quote A. Lenstra, H. Lenstra, Manasse, and Pollard [37]: "There is no indication that the modification proposed by Coppersmith has any practical value."

At the time there was already more than a decade of literature showing how to analyze algorithm asymptotics in more realistic models of computation that account for memory consumption, communication, etc.; see, e.g., [18]. Bernstein in [7] analyzed the circuit performance of NFS, concluding that an optimized circuit of area $L^{0.790\ldots+o(1)}$ would factor N in time $L^{1.18\ldots+o(1)}$, for price-performance ratio $L^{1.976\ldots+o(1)}$. [7] did not analyze the factorization factory but did analyze multiple number fields, concluding that they did not reduce AT cost. The gap between the RAM exponent $1.901\ldots + o(1)$ from [20] and the AT exponent $1.976\ldots + o(1)$ from [7] is explained primarily by communication overhead inside linear algebra, somewhat moderated by parameter choices that reduce the cost of linear algebra at the expense of relation collection.

We pointed out in [14] that the factorization factory does not reduce AT cost. In Section 2 we review the reason for this and explain how batch NFS works around it. We also presented in [14] a superpolynomial improvement to the factorization factory in the RAM metric, by eliminating ECM in favor of batch trial division, but this is not useful in the AT metric.

2 Exponents

This section reviews NFS and then explains how to drastically reduce the AT cost of NFS through batching. The resulting cost exponent, $1.704\ldots$ in Table 1.4, is new. All costs in this section are expressed as $L^{e+o(1)}$ for various exponents e. Section 3 looks more closely at the $L^{o(1)}$ factor.

2.1. QS: the quadratic sieve (1982). As a warmup for NFS we briefly review the general idea of combining congruences, using QS as an example.

QS writes down a large collection of congruences modulo the target integer N and tries to find a nontrivial subcollection whose product is a congruence of squares. One can then reasonably hope that the difference of square roots has a nontrivial factor in common with N.

Specifically, QS computes $s \approx \sqrt{N}$ and writes down the congruences $s^2 \equiv s^2 - N$, $(s+1)^2 \equiv (s+1)^2 - N$, etc. The left side of each congruence is already a square. The main problem is to find a nontrivial set of integers a such that the product of $(s+a)^2 - N$ is a square.

If $(s+a)^2 - N$ is divisible by a very large prime then it is highly unlikely to participate in a square: the prime would have to appear a second time. QS therefore focuses on **smooth** congruences: congruences where $(s+a)^2-N$ factors completely into small primes. Applying linear algebra modulo 2 to the matrix of exponents in these factorizations is guaranteed to find nonempty subsets of the congruences with square product once the number of smooth congruences exceeds the number of small primes.

The integers a such that $(s+a)^2 - N$ is divisible by a prime p form a small number of arithmetic progressions modulo p. "Sieving" means jumping through these arithmetic progressions to mark divisibility, the same way that the sieve of Eratosthenes jumps through arithmetic progressions to mark non-primality.

2.2. NFS: the number-field sieve (1993). NFS applies the same idea, but instead of congruences modulo N it uses congruences modulo a related algebraic number $m - \alpha$. This algebraic number is chosen to have norm N (divided by a certain denominator shown below), and one can reasonably hope to obtain a factorization of N by obtaining a random factorization of this algebraic number.

Specifically, NFS chooses a positive integer m, and writes N as a polynomial in radix m, namely $N = f(m)$ where f is a degree-d polynomial with coefficients $f_d, f_{d-1}, \ldots, f_0 \in \{0, 1, \ldots, m-1\}$. It is not difficult to see that optimizing NFS requires d to grow slowly with N, so m is asymptotically on a much smaller scale than N, although not as small as L. More precisely, NFS takes

$$m \in \exp((\mu + o(1))(\log N)^{2/3}(\log\log N)^{1/3})$$

where μ is a positive real constant, optimized below. Note that the inequalities $m^d \le N < m^{d+1}$ imply

$$d \in (1/\mu + o(1))(\log N)^{1/3}(\log\log N)^{-1/3}.$$

If f is reducible then its factorization is easy to compute and (for N reasonably large compared to d) reveals a nontrivial factorization of N (see [36]), so assume from now on that f is irreducible. Define α as a root of f. The norm of $a - b\alpha$ is then $f_d a^d + f_{d-1} a^{d-1} b + \cdots + f_0 b^d$ (divided by f_d), and in particular the norm of $m - \alpha$ is N (again divided by f_d).

NFS uses the congruences $a - bm \equiv a - b\alpha$ modulo $m - \alpha$. There are now two numbers, $a - bm$ and $a - b\alpha$, that both need to be smooth. Smoothness of the algebraic number $a - b\alpha$ is defined as smoothness of the (scaled) norm $f_d a^d + f_{d-1} a^{d-1} b + \cdots + f_0 b^d$, and smoothness of an integer is defined as having no prime divisors larger than y. Here $y \in L^{\gamma+o(1)}$ is another parameter chosen by NFS; $\gamma > 1/(6\mu)$ is another real constant, optimized below.

The range of pairs (a, b) searched for smooth congruences is the set of coprime integer pairs in the rectangle $[-H, H] \times [1, H]$. Here H is chosen so that there will be enough smooth congruences to produce squares at the end of the algorithm. Standard heuristics state that $a - bm$ has smoothness probability $L^{-\mu/(3\gamma)+o(1)}$ if a and b are on much smaller scales than m; in particular, if $H \in L^{\theta+o(1)}$ for some positive real number θ then the number of congruences with $a - bm$ smooth is $L^{\phi+o(1)}$ with $\phi = 2\theta - \mu/(3\gamma)$. Standard heuristics also provide the simultaneous smoothness probability of $a - bm$ and $a - b\alpha$, implying that to obtain enough smooth congruences one can take $H \in L^{\theta+o(1)}$ with $\theta = (3\mu\gamma^2 + 2\mu^2)/(6\mu\gamma - 1)$ and $\phi = (18\mu\gamma^3 + 6\mu^2\gamma + \mu)/(18\mu\gamma^2 - 3\gamma)$. See, e.g., [19]. We henceforth assume these formulas for θ and ϕ in terms of μ and γ.

2.3. RAM cost analysis (1993). Sieving for y-smoothness of $H^{2+o(1)}$ polynomial values uses $H^{2+o(1)}$ operations, provided that y is bounded by $H^{2+o(1)}$. The point here is that the pairs (a, b) with congruences divisible by p form a small number of shifted lattices of determinant p, usually with basis vectors of length $O(\sqrt{p})$, making it easy to find all the lattice points inside the rectangle $[-H, H] \times [1, H]$. The number of operations is thus essentially the number of points marked, and each point is marked just $\sum_{p \le y} 1/p \approx \log\log y$ times.

Sparse techniques for linear algebra involve $y^{1+o(1)}$ matrix-vector multiplications, each involving $y^{1+o(1)}$ operations, for a total of $y^{2+o(1)}$ operations. Other subroutines in NFS take negligible time, so the overall RAM cost of NFS is $L^{\max\{2\theta, 2\gamma\}+o(1)}$.

It is not difficult to see that the exponent $\max\{2\theta, 2\gamma\}$ achieves its minimum value $(64/9)^{1/3} = 1.922\ldots$ with $\mu = (1/3)^{1/3} = 0.693\ldots$ and $\theta = \gamma = (8/9)^{1/3} = 0.961\ldots$. This exponent $1.922\ldots$ is the NFS exponent from [19], and as mentioned earlier is the most frequently quoted NFS exponent. We do not review the multiple-number-fields improvement to $1.901\ldots$ from [20]; as far as we know, multiple number fields do not improve any of the exponents analyzed below.

2.4. *AT* cost analysis (2001). In the AT metric there is an important obstacle to cost $H^{2+o(1)}$ for sieving: namely, communicating across area $H^{2+o(1)}$ takes time at least $H^{1+o(1)}$. One can efficiently split the sieving problem into $H^{2+o(1)}/y^{1+o(1)}$ tasks, running one task after another on a smaller array of size $y^{1+o(1)}$, but communicating across this array still takes time at least $y^{0.5+o(1)}$, so AT is at least $H^{2+o(1)} y^{0.5+o(1)}$.

Fortunately, there is a much more efficient alternative to sieving: ECM, explained in Appendix A. What matters in this section is that ECM tests y-smoothness in time $y^{o(1)}$ on a circuit of area $y^{o(1)}$. A parallel array of ECM units, each handling a separate number, tests y-smoothness of $H^{2+o(1)}$ polynomial values in time $H^{2+o(1)}/y^{1+o(1)}$ on a circuit of area $y^{1+o(1)}$, achieving $AT = H^{2+o(1)}$.

Unfortunately, the same obstacle shows up again for linear algebra, and this time there is no efficient alternative. Multiplying a sparse matrix by a vector requires time $y^{0.5+o(1)}$ on a circuit of area $y^{1+o(1)}$, and must be repeated $y^{1+o(1)}$ times. The overall AT cost of NFS is $L^{\max\{2\theta,2.5\gamma\}+o(1)}$.

The exponent $\max\{2\theta, 2.5\gamma\}$ achieves its minimum value $1.976\ldots$ with $\mu = 0.702\ldots$, $\gamma = 0.790\ldots$, and $\theta = 0.988\ldots$. This exponent $1.976\ldots$ is the NFS exponent from [7]. Notice that γ is much smaller here than it was in the RAM optimization: y has been reduced to keep the cost of linear algebra under control, but this also forced θ to increase.

2.5. The factorization factory (1993). Coppersmith in [20] precomputes "tables which will be useful for factoring any integers in a large range ... after the precomputation, an individual integer can be factored in time $L[1/3, 1.639]$", i.e., $L^{\approx 1.639+o(1)}$.

Coppersmith's table is simply the set of (a, b) such that $a-bm$ is smooth. One reuses m, and thus this table, for any integer N between (e.g.) m^d and m^{d+1}.

Coppersmith's method to factor "an individual integer" is to test smoothness of $a - b\alpha$ for each (a, b) in the table. At this point Coppersmith has found the same smooth congruences as conventional NFS, and continues with linear algebra in the usual way.

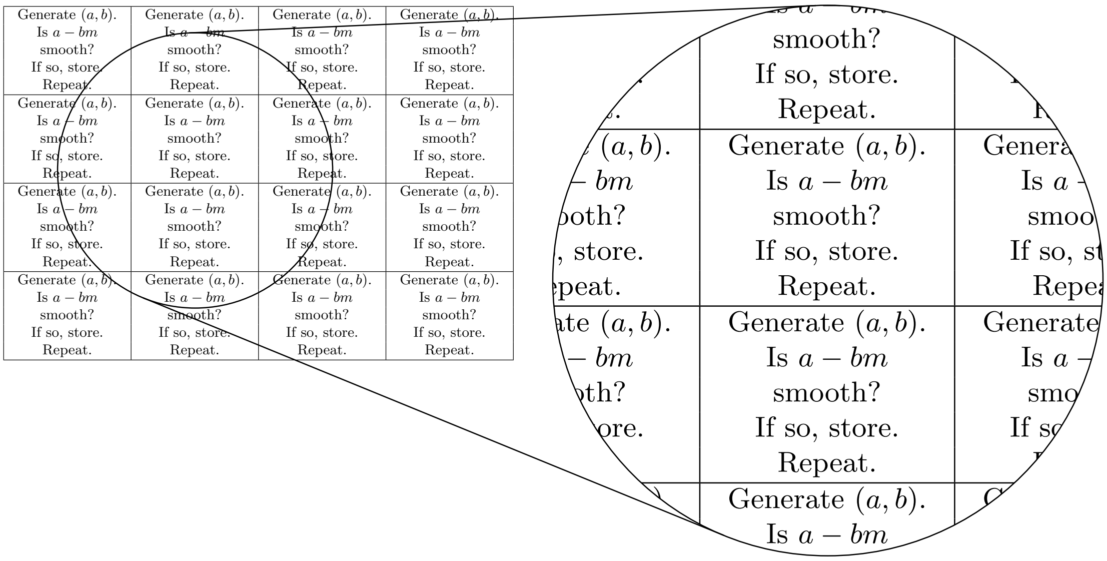
Coppersmith uses ECM to test smoothness. The problem with sieving here is not efficiency, as in the (subsequent) paper [7], but functionality: sieving can handle polynomial values only at regularly spaced inputs, and the pairs (a, b) in this table are not regularly spaced.

Recall that the size of this table is $L^{\phi+o(1)}$ with $\phi = 2\theta - \mu/(3\gamma)$. ECM uses $L^{o(1)}$ operations per number, for a total smoothness cost of $L^{\phi+o(1)}$, asymptotically a clear improvement over the $L^{2\theta+o(1)}$ for conventional NFS.

The overall RAM cost of the factorization factory is $L^{\max\{\phi,2\gamma\}+o(1)}$. The exponent achieves its minimum value $1.638\ldots$ with $\mu = 0.905\ldots$, $\gamma = 0.819\ldots$, $\theta = 1.003\ldots$, and $\phi = 1.638\ldots$. This is the exponent from [20].

The AT metric tells a completely different story, as we pointed out in [14]. The area required for the table is $L^{\phi+o(1)}$. This area is easy to reuse for very fast parallel smoothness detection, finishing in time $L^{o(1)}$. Unfortunately, collecting the smooth results then takes time $L^{0.5\phi+o(1)}$, for an AT cost of at least $L^{\max\{1.5\phi,2.5\gamma\}+o(1)}$, never mind the problem of matching the table area with the linear-algebra area. The minimum exponent here is above 2.4.

2.6. Batch NFS (new). We drastically reduce AT cost by sharing work across many N's in a different way: we process a *batch* of N's in parallel, rather than performing precomputation to be used for one N at a time. We dynamically enumerate the pairs (a, b) with $a - bm$ smooth, distribute each pair across all



Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.
Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.
Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.
Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.	Generate (a,b). Is $a-bm$ smooth? If so, store. Repeat.

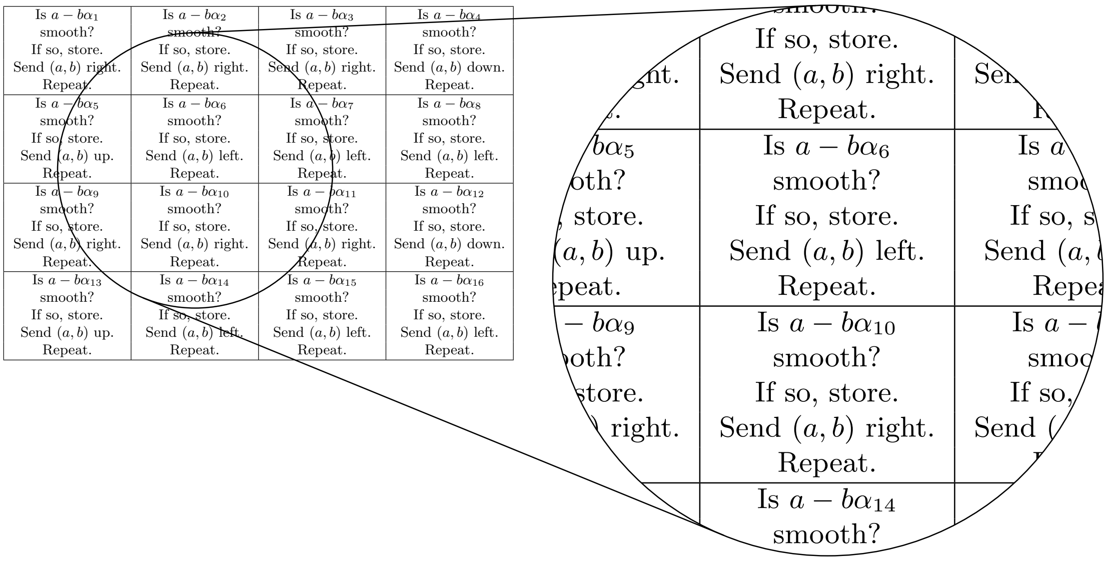
Fig. 2.7. Relation-search mesh finding pairs (a,b) where $a-bm$ is smooth. The following exponents are optimized for factoring a batch of $L^{0.5+o(1)}$ B-bit integers: The mesh has height $L^{0.25+o(1)}$, width $L^{0.25+o(1)}$, and area $L^{0.5+o(1)}$. The mesh consists of $L^{0.5+o(1)}$ small parallel processors (illustration contains 16). Each processor has area $L^{o(1)}$. Each processor knows the same $m \in \exp((0.92115 + o(1))(\log 2^B)^{2/3}(\log\log 2^B)^{1/3})$. Each processor generates its own $L^{0.200484+o(1)}$ pairs (a,b), where a and b are bounded by $L^{1.077242+o(1)}$. Each processor tests each of its own $a-bm$ for smoothness using ECM, using smoothness bound $L^{0.681600+o(1)}$. Together the processors generate $L^{0.700484+o(1)}$ separate pairs (a,b), of which $L^{0.25+o(1)}$ have $a-bm$ smooth.

the N's in the batch, and remove each pair as soon as possible, rather than storing a complete table of the pairs. To avoid excessive communication costs we completely reorganize data in the middle of the computation: at the beginning each N is repeated many times to bring N close to the pairs (a,b), while at the end the pairs (a,b) relevant to each N are moved much closer together. The rest of this subsection presents the details of the algorithm.

Consider as input a batch of $L^{\beta+o(1)}$ simultaneous targets N within the large range described above. We require $\beta \le \min\{2\phi-2\gamma, 4\theta-2\phi\}$; if there are more targets available at once then we actually process those targets in batches of size $L^{\min\{2\phi-2\gamma,4\theta-2\phi\}+o(1)}$, storing no data between runs.

Consider a square mesh of $L^{\beta+o(1)}$ small parallel processors. This mesh is large enough to store all of the targets N. Use each processor in parallel to test smoothness of $a-bm$ for $L^{2\theta-\phi-0.5\beta+o(1)}$ pairs (a,b) using ECM; by hypothesis $2\theta-\phi-0.5\beta \ge 0$. The total number of pairs here is $L^{2\theta-\phi+0.5\beta+o(1)}$. Each smoothness test takes time $L^{o(1)}$. Overall the mesh takes time $L^{2\theta-\phi-0.5\beta+o(1)}$ and produces a total of $L^{0.5\beta+o(1)}$ pairs (a,b) with $a-bm$ smooth, i.e., only $L^{o(1)}$ pairs for each column of the mesh. See Figure 2.7.

Move these pairs to the top row of the mesh (spreading them evenly across that row) by a standard sorting algorithm, say the Schnorr–Shamir algorithm



Is $a - b\alpha_1$ smooth? If so, store. Send (a,b) right. Repeat.	Is $a - b\alpha_2$ smooth? If so, store. Send (a,b) right. Repeat.	Is $a - b\alpha_3$ smooth? If so, store. Send (a,b) right. Repeat.	Is $a - b\alpha_4$ smooth? If so, store. Send (a,b) down. Repeat.
Is $a - b\alpha_5$ smooth? If so, store. Send (a,b) up. Repeat.	Is $a - b\alpha_6$ smooth? If so, store. Send (a,b) left. Repeat.	Is $a - b\alpha_7$ smooth? If so, store. Send (a,b) left. Repeat.	Is $a - b\alpha_8$ smooth? If so, store. Send (a,b) left. Repeat.
Is $a - b\alpha_9$ smooth? If so, store. Send (a,b) right. Repeat.	Is $a - b\alpha_{10}$ smooth? If so, store. Send (a,b) right. Repeat.	Is $a - b\alpha_{11}$ smooth? If so, store. Send (a,b) right. Repeat.	Is $a - b\alpha_{12}$ smooth? If so, store. Send (a,b) down. Repeat.
Is $a - b\alpha_{13}$ smooth? If so, store. Send (a,b) up. Repeat.	Is $a - b\alpha_{14}$ smooth? If so, store. Send (a,b) left. Repeat.	Is $a - b\alpha_{15}$ smooth? If so, store. Send (a,b) left. Repeat.	Is $a - b\alpha_{16}$ smooth? If so, store. Send (a,b) left. Repeat.

Fig. 2.8. Relation-search mesh from Figure 2.7, now finding pairs (a, b) where both $a - bm$ and $a - b\alpha_i$ are smooth. For a batch of $L^{0.5+o(1)}$ B-bit integers: The mesh knows $L^{0.25+o(1)}$ pairs (a, b) with $a - bm$ smooth from Figure 2.7. Each (a, b) is copied $L^{0.25+o(1)}$ times (2 times in the illustration) so that it appears in the first two rows, the next two rows, etc. Each (a, b) visits each mesh position within $L^{0.25+o(1)}$ steps (8 steps in the illustration). Each processor knows its own target N_i and the corresponding α_i, and in each step tests each $a - b\alpha_i$ for smoothness using ECM. Together Figure 2.7 and Figure 2.8 take time $L^{0.25+o(1)}$ to search $L^{0.700484+o(1)}$ pairs (a, b).

from [50], taking time $L^{0.5\beta+o(1)}$. Then broadcast each pair to its entire column, taking time $L^{0.5\beta+o(1)}$. Actually, it will suffice for each pair to appear once somewhere in the first two rows, once somewhere in the next two rows, etc.

Now consider a pair at the top-left corner. Send this pair to its right until it reaches the rightmost column, then down one row, then repeatedly to its left, then back up. In parallel move all the other elements in the first two rows on the same path. In parallel do the same for the third and fourth rows, the fifth and sixth rows, etc. Overall this takes time $L^{0.5\beta+o(1)}$.

Observe that each pair has now visited each position in the mesh. When a pair (a, b) visits a mesh position holding a target N, use ECM to check whether $a - b\alpha$ is smooth, taking time $L^{o(1)}$. The total time to check all $L^{0.5\beta+o(1)}$ pairs against all $L^{\beta+o(1)}$ targets is just $L^{0.5\beta+o(1)}$, plus the time $L^{2\theta-\phi-0.5\beta+o(1)}$ to generate the pairs in the first place. See Figure 2.8.

Repeat this entire procedure $L^{\phi-\gamma-0.5\beta+o(1)}$ times; by hypothesis $\phi - \gamma - 0.5\beta \geq 0$. This covers a total of $L^{2\theta-\gamma+o(1)}$ pairs (a, b), of which $L^{\phi-\gamma+o(1)}$ have $a - bm$ smooth, so for each N there are $L^{o(1)}$ pairs (a, b) for which $a - bm$ and $a - b\alpha$ are both smooth. The total number of smooth congruences found this way across all N is $L^{\beta+o(1)}$. Store each smooth congruence as (N, a, b); all of these together fit into a mesh of area $L^{\beta+o(1)}$. The time spent is $L^{\max\{\phi-\gamma,2\theta-\gamma-\beta\}+o(1)}$.

Build $L^{\gamma+o(1)}$ copies of the same mesh, all operating in parallel, for a total circuit area of $L^{\beta+\gamma+o(1)}$. Each copy of the mesh has its own copy of the entire

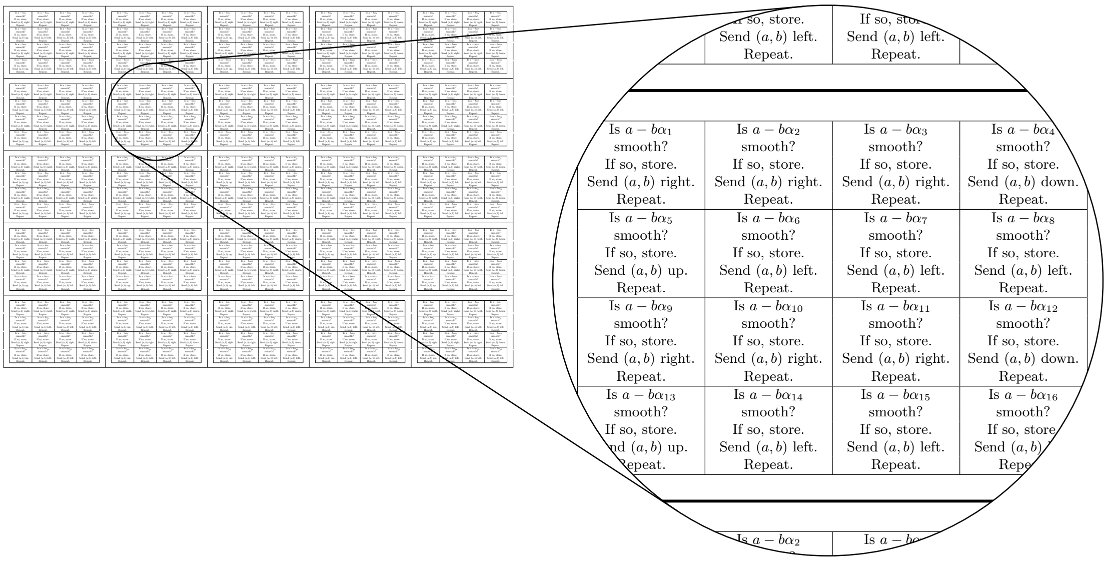

so, store.
Send (a,b) left.
Repeat.
If so, sto
Send (a,b) left.
Repeat.
Is $a-b\alpha_1$ smooth? If so, store. Send (a,b) right. Repeat.
Is $a-b\alpha_2$ smooth? If so, store. Send (a,b) right. Repeat.
Is $a-b\alpha_3$ smooth? If so, store. Send (a,b) right. Repeat.
Is $a-b\alpha_4$ smooth? If so, store. Send (a,b) down. Repeat.
Is $a-b\alpha_5$ smooth? If so, store. Send (a,b) up. Repeat.
Is $a-b\alpha_6$ smooth? If so, store. Send (a,b) left. Repeat.
Is $a-b\alpha_7$ smooth? If so, store. Send (a,b) left. Repeat.
Is $a-b\alpha_8$ smooth? If so, store. Send (a,b) left. Repeat.
Is $a-b\alpha_9$ smooth? If so, store. Send (a,b) right. Repeat.
Is $a-b\alpha_{10}$ smooth? If so, store. Send (a,b) right. Repeat.
Is $a-b\alpha_{11}$ smooth? If so, store. Send (a,b) right. Repeat.
Is $a-b\alpha_{12}$ smooth? If so, store. Send (a,b) down. Repeat.
Is $a-b\alpha_{13}$ smooth? If so, store. nd (a,b) up. epeat.
Is $a-b\alpha_{14}$ smooth? If so, store. Send (a,b) left. Repeat.
Is $a-b\alpha_{15}$ smooth? If so, store. Send (a,b) left. Repeat.
Is $a-b\alpha_{16}$ smooth? If so, store Send (a,b) Repe
Is $a-b\alpha_2$
Is $a-b$

Fig. 2.9. For a batch of $L^{0.5+o(1)}$ B-bit integers: $L^{0.681600+o(1)}$ copies (25 copies in the illustration) of the mesh from Figure 2.7 and Figure 2.8. Each copy has the same $L^{0.5+o(1)}$ target integers to factor. The total area of this circuit is $L^{1.181600+o(1)}$. In time $L^{0.25+o(1)}$ this circuit searches $L^{1.382084+o(1)}$ pairs (a,b). In time $L^{1.022400+o(1)}$ this circuit searches all $L^{2.154484+o(1)}$ pairs (a,b) and finds, for each target N_i and the corresponding α_i, all $L^{0.681600+o(1)}$ pairs (a,b) for which $a-bm$ and $a-b\alpha_i$ are both smooth.

list of N's; distributing the N's from an input port through the total circuit area takes time $L^{0.5\beta+0.5\gamma+o(1)}$. The total circuit covers all $L^{2\theta+o(1)}$ pairs (a,b) and obtains, for each N, all of the $L^{\gamma+o(1)}$ smooth congruences required to factor that N. See Figure 2.9.

We are not done yet: we still need to perform linear algebra for each N. To keep the communication costs of linear algebra under control we pack the linear algebra for each N into the smallest possible area. Allocate a separate square of area $L^{\gamma+o(1)}$ to each N, and route each smooth congruence (N,a,b) in parallel to the corresponding square; this is another standard sorting step, taking total time $L^{0.5\beta+0.5\gamma+o(1)}$ for all $L^{\beta+\gamma+o(1)}$ smooth congruences. Finally, perform linear algebra separately in each square, and complete the factorization of each N as usual. This takes time $L^{1.5\gamma+o(1)}$. See Figure 2.10.

The overall time exponent is $\max\{\phi-\gamma, 2\theta-\gamma-\beta, 0.5\beta+0.5\gamma, 1.5\gamma\}$, and the area exponent is $\beta+\gamma$. The final price-performance ratio, AT per integer factored, has exponent $\max\{\phi, 2\theta-\beta, 0.5\beta+1.5\gamma, 2.5\gamma\}$.

2.11. Comparison and numerical parameter optimization. Bernstein's AT exponent from [7] was $\max\{2\theta, 2.5\gamma\}$. Batch NFS replaces 2θ with $2\theta-\beta$, allowing γ to be correspondingly reduced, at least until β becomes large enough for 2.5γ to cross below ϕ. In principle one should also watch for 2.5γ to cross below $0.5\beta+1.5\gamma$, but Table 2.12 shows that ϕ is more important.

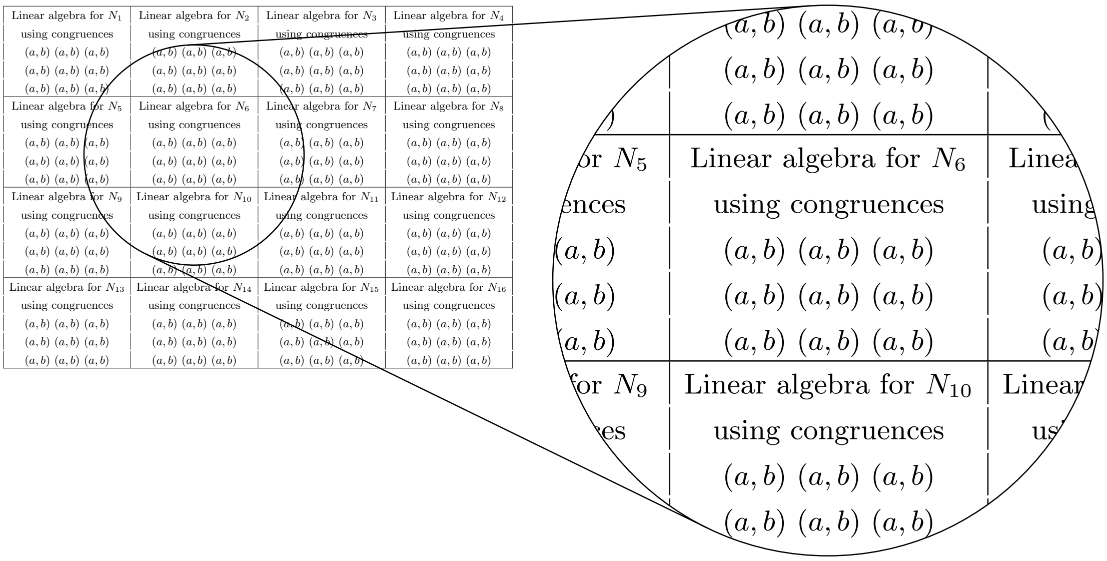


Fig. 2.10. For a batch of $L^{0.5+o(1)}$ B-bit integers: $L^{0.5+o(1)}$ copies (16 copies in the illustration) of a linear-algebra circuit. Each circuit has area $L^{0.681600+o(1)}$. The total area is $L^{1.181600+o(1)}$. Each circuit has its own integer N_i to factor and $L^{0.681600+o(1)}$ pairs (a,b) for which $a-bm$ and $a-b\alpha_i$ are smooth. Routing all pairs (a,b) from Figure 2.9 to an adjacent (or overlapping and reconfigured) Figure 2.10 takes time $L^{0.590800+o(1)}$. Each circuit uses $L^{0.681600+o(1)}$ matrix-vector multiplications, and takes time $L^{0.340800+o(1)}$ for each matrix-vector multiplication. The total time is $L^{1.022400+o(1)}$.

Of course, even if we ignore the cost of finding the smooth $a-bm$ (the term $2\theta-\beta$), our AT exponent is not as small as Coppersmith's RAM exponent $\max\{\phi,2\gamma\}$ from [20]. We have an extra $0.5\beta+1.5\gamma$ term, reflecting the cost of communicating smooth congruences across a batch, and, more importantly, 2.5γ instead of 2γ, reflecting the communication cost of linear algebra.

Table 2.12 shows the smallest exponents that we obtained for various β, in each case from a brief search through 2500000000 pairs (μ,γ). The exponent of the price-performance ratio for batch NFS drops below Bernstein's $1.976\ldots$ as soon as β increases past 0, and reaches a minimum of $1.704\ldots$ as the batch size increases. (The minimum is actually very slightly below 1.704, but our table does not include enough precision to show this.) Finding all (a,b) with $a-bm$ smooth is still a slight bottleneck for $\beta=0.4$ but disappears for $\beta=0.5$. When there are more inputs we partition them into batches of size $L^{0.5+o(1)}$, preserving exponent $1.704\ldots$ for the price-performance ratio.

Our optimal $\gamma=0.681\ldots$ is much smaller than Coppersmith's $\gamma=0.819\ldots$, for the same reasons that Bernstein's $\gamma=0.790\ldots$ is smaller than the conventional $\gamma=0.961\ldots$. The natural time exponent for batch NFS — as above, this means the time exponent when price-performance ratio is optimized — is just $1.022\ldots$, considerably smaller than the natural time exponent $1.185\ldots$ for single-key NFS. This means that collecting targets into batches produces not merely a drastic improvement in price-performance ratio, but also a side effect of considerably reducing latency.

batch	AT	m size	primes	(a, b)	$a - b\alpha$	$a - bm$	route	linear
β	e	μ	γ	θ	ϕ	$2\theta - \beta$	$0.5\beta + 1.5\gamma$	2.5γ
0.0	1.976052	0.702860	0.790420	0.988026	1.679645	1.976052	1.185630	1.976050
0.1	1.900575	0.705460	0.760230	1.000287	1.691256	1.900575	1.190345	1.900575
0.2	1.829615	0.712320	0.731840	1.014808	1.705173	1.829615	1.197760	1.829600
0.3	1.763034	0.718160	0.705210	1.031517	1.723580	1.763034	1.207815	1.763025
0.4	1.710375	0.820920	0.684150	1.055172	1.710374	1.710345	1.226225	1.710375
0.5	1.704000	0.921150	0.681600	1.077242	1.704000	1.654484	1.272400	1.704000

Table 2.12. Cost exponents for batch NFS in the AT metric. The batch size is $L^{\beta+o(1)}$. The AT cost is $L^{e+o(1)}$. The parameter m is chosen as $\exp((\mu + o(1))(\log N)^{2/3}(\log\log N)^{1/3})$. The prime bound y is chosen as $L^{\gamma+o(1)}$. The (a, b) bound H is chosen as $L^{\theta+o(1)}$. The number of $a - b\alpha$ smoothness tests is $L^{\phi+o(1)}$ per target. The number of $a - bm$ smoothness tests is $L^{2\theta-\beta+o(1)}$ per target. The AT cost of routing is $L^{0.5\beta+1.5\gamma+o(1)}$ per target. The AT cost of linear algebra is $L^{2.5\gamma+o(1)}$ per target. All operations take place on a circuit of size $L^{\beta+\gamma+o(1)}$.

3 Early-abort ECM

Section 2 used ECM as a low-area smoothness test for auxiliary integers $c = f_d a^d + \cdots + f_0 b^d$. Each curve in ECM catches a fraction of the primes $p \leq y$ dividing c, and many curves in sequence catch essentially all of the primes $p \leq y$.

This section analyzes a much faster smoothness-detection method, "early-abort ECM". Not all smooth numbers are detected by early-abort ECM, but new heuristics introduced in this section imply that this loss is much smaller than the speedup factor. The overall improvement grows as a superpolynomial function of $\log y$, and therefore grows as a superpolynomial function of the NFS input size.

Specifically, it is well known (see, e.g. [21, page 302]) that (assuming standard conjectures) ECM uses $\exp(\sqrt{(2 + o(1))\log y \log\log y})$ multiplications modulo c to find essentially all primes $p \leq y$ dividing c. Here $o(1)$ is some function of y that converges to 0 as $y \to \infty$. Consequently, if a fraction $1/S$ of the ECM inputs are smooth, then ECM uses

$$S \cdot \exp(\sqrt{(2 + o(1))\log y \log\log y})$$

modular multiplications for each smooth integer that it finds. This section's heuristics imply that early-abort ECM uses only

$$S \cdot \exp\left(\sqrt{\left(\frac{8}{9} + o(1)\right) \log y \log\log y}\right)$$

modular multiplications for each smooth integer that it finds. Notice the change from $2 + o(1)$ to $8/9 + o(1)$ in the exponent.

We emphasize again that this paper's analyses are asymptotic. We do not claim that early-abort ECM is better than ECM for any particular value of y.

The rest of this section uses the word “time” to count simple arithmetic operations, such as multiplication and division, on integers with $O(\lg c)$ bits. Each of these operations actually takes time $(\lg c)^{1+o(1)}$, but this extra factor is absorbed into other $o(1)$ terms when c is bounded by the usual functions of y.

3.1. Early-abort trial division. Early aborts predate ECM. They became popular in the 1970s as a component of CFRAC [43], a subexponential-time factorization method that, like batch NFS, generates many “random” numbers that need to be tested for smoothness.

The simplest form of early aborts is single-early-abort trial division. Trial division simply checks divisibility of c by each prime $p \le y$, taking time $y^{1+o(1)}$. Single-early-abort trial division first checks divisibility of c by each prime $p \le y^{1/2}$; then throws c away (this is the early abort) if the unfactored part of c is too large; and then, if c has survived the early abort, checks divisibility of c by each prime $p \le y$. (Of course, when checking each prime $p \le y$, one can skip the redundant checks of primes $p \le y^{1/2}$.)

The definition of “too large” is chosen so that $1/y^{1/2+o(1)}$ of all inputs survive the abort, balancing the cost of the stages before and after the abort. In other words, single-early-abort trial division checks divisibility of each input by each prime $p \le \sqrt{y}$; keeps the smallest $1/y^{1/2+o(1)}$ of all inputs; and, for each of those inputs, checks divisibility by each prime $p \le y$.

More generally, $(k-1)$-early-abort trial division removes each prime $p \le y^{1/k}$ from each input (by dividing by factors found); reduces the number of inputs by a factor of $y^{1/k}$, keeping the smallest inputs; removes each prime $p \le y^{2/k}$ from each remaining input; reduces the number of inputs by another factor of $y^{1/k}$, keeping the smallest inputs; and so on through $y^{k/k} = y$.

The time per input for $(k-1)$-early-abort trial division is only $y^{1/k+o(1)}$, saving a factor $y^{1-1/k+o(1)}$, if k is limited to a slowly growing function of y. The method does not detect all smooth numbers, but Pomerance’s analysis in [45, Section 4] shows that the loss factor is only $y^{(1-1/k)/2+o(1)}$, i.e., that the method detects 1 out of every $y^{(1-1/k)/2+o(1)}$ smooth numbers. The overall improvement factor in price-performance ratio is $y^{(1-1/k)/2+o(1)}$; if k is chosen so that $k \to \infty$ as $y \to \infty$ then the improvement factor is $y^{1/2+o(1)}$.

3.2. Early aborts in more generality. One can replace trial division with any method, or combination of methods, of checking for primes $\le y^{1/k}$, primes $\le y^{2/k}$, etc.

In particular, Pomerance considered an early-abort version of Pollard’s rho method. The original method takes time $y^{1/2+o(1)}$ to find all primes $p \le y$. Early-abort rho takes time only $y^{1/(2k)+o(1)}$, and Pomerance’s analysis shows that it has a loss factor of only $y^{(1-1/k)/4+o(1)}$.

Pomerance actually considered a different method by Pollard and Strassen. The Pollard–Strassen method takes essentially the same amount of time as Pollard’s rho method, and has the advantage of a proof of speed without any conjectures, but has the disadvantage of using much more memory.

Pomerance’s paper was published in 1982, so of course it did not analyze the elliptic-curve method. After seeing early aborts improve trial division from

y to $y^{1/2}$, and improve Pollard's rho method from $y^{1/2}$ to $y^{1/4}$, one might guess that early aborts improve ECM from $\exp(\sqrt{(2+o(1))\log y \log\log y})$ to $\exp((1/2)\sqrt{(2+o(1))\log y \log\log y})$, but our heuristics do not agree with this guess.

3.3. Performance of early aborts. Recall that ECM takes time $T(y)^{1+o(1)}$ to find primes $p \le y$, where $T(y) = \exp(\sqrt{2\log y \log\log y})$. We actually consider, in much more generality, any factorization method M taking time $T(y)^{1+o(1)}$ to find primes $p \le y$, where T is any sufficiently smooth function.

Our early-abort heuristics state that the price-performance ratio of $(k-1)$-early-abort M is the geometric average

$$T(y^{1/k})^{1/k}T(y^{2/k})^{1/k}T(y^{3/k})^{1/k}\cdots T(y)^{1/k}$$

to the power $1+o(1)$. More generally, cutoffs $y_1, y_2, y_3, \ldots$ produce a geometric average of $T(y_1), T(y_2), T(y_3), \ldots$ with weights $\log y_1, \log y_2 - \log y_1, \log y_3 - \log y_2, \ldots$.

In particular, for any purely exponential $T(y) = y^C$, the price-performance ratio is (aside from the $1+o(1)$ power)

$$\begin{aligned}\left(T(y^{1/k})T(y^{2/k})\cdots T(y^{(k-1)/k})T(y)\right)^{1/k} &= \left(y^{C/k}y^{2C/k}\cdots y^{(k-1)C/k}y^{C}\right)^{1/k}\\ &= \left(y^{C\sum_{i=1}^{k} i/k}\right)^{1/k} = y^{C(k+1)/(2k)}\end{aligned}$$

which converges to $y^{C/2} = T(y)^{1/2}$ as k increases, matching Pomerance's analyses of early-abort trial division and early-abort rho. More generally, if $T(y) = \exp(C(\log y)^{1/f})$ then $T(y^{i/k}) = T(y)^{(i/k)^{1/f}}$ so

$$\left(T(y^{1/k})T(y^{2/k})\cdots T(y^{(k-1)/k})T(y)\right)^{1/k} = T(y)^{(\sum_{i=1}^{k}(i/k)^{1/f})/k} \to T(y)^{f/(f+1)}.$$

To prove that $(\sum_{i=1}^{k}(i/k)^{1/f})/k \to f/(f+1)$ as $k \to \infty$, observe that $\sum_{i=1}^{k} i^{1/f}$ is within $k^{1/f}$ of $\int_0^k z^{1/f}\,dz = (f/(f+1))k^{(f+1)/f}$. ECM is essentially the case $f = 2$: the geometric average is $T(y)^{2/3+o(1)}$.

3.4. Understanding the heuristics. Let y and u be real numbers larger than 1, define $x = y^u$, and define $S_0 = \{1, 2, \ldots, \lfloor x \rfloor\}$. Define $\Psi(x, y)$ as the number of y-smooth integers in S_0. Then $\Psi(x, y)$ is approximately x/u^u. See [45, Theorem 2.1] for a precise statement. The same approximation is still valid for $\Psi(x, y, z)$, the number of y-smooth integers in S_0 having no prime factor $\le z$, assuming that $z < y^{1-1/\log u}$; see [45, Theorem 2.2].

Let k be a positive integer. Let $y_0, y_1, y_2, \ldots, y_k$ be real numbers with $1 = y_0 < y_1 < y_2 < \cdots < y_k = y$. Let $x_1, x_2, \ldots, x_k$ be positive real numbers with

$x = x_1 x_2 \cdots x_k$. Define

$$\begin{aligned} S_1 &= \{c \in S_0 : c/(y_1\text{-smooth part of } c) \le x/x_1\}; \\ S_2 &= \{c \in S_1 : c/(y_2\text{-smooth part of } c) \le x/(x_1 x_2)\}; \\ &\vdots \\ S_k &= \{c \in S_{k-1} : c/(y_k\text{-smooth part of } c) \le x/(x_1 x_2 \cdots x_k)\}. \end{aligned}$$

Note that each element $c \in S_k$ is y-smooth, since c divided by its y-smooth part is bounded by $x/(x_1 x_2 \cdots x_k) = 1$.

Consider any vector $(s_1, s_2, \ldots, s_k)$ such that each s_i is a y_i-smooth positive integer $\le x_i$ having no prime factors $\le y_{i-1}$. For any such $(s_1, s_2, \ldots, s_k)$, the product $c = s_1 s_2 \cdots s_k$ is a positive integer bounded by $x_1 x_2 \cdots x_k = x$, so $c \in S_0$. Dividing c by its y_1-smooth part produces $s_2 \cdots s_k \le x/x_1$, so $c \in S_1$. Similarly $c \in S_2$ and so on through $c \in S_k$.

The map from $(s_1, s_2, \ldots, s_k)$ to $s_1 s_2 \cdots s_k \in S_k$ is injective: the y_1-smooth part of $s_1 s_2 \cdots s_k$ is exactly s_1, the y_2-smooth part is exactly $s_1 s_2$, etc. Hence $\#S_k$ is at least the number of such vectors $(s_1, s_2, \ldots, s_k)$, which is exactly $\Psi(x_1, y_1, y_0)\Psi(x_2, y_2, y_1)\Psi(x_3, y_3, y_2) \cdots \Psi(x_k, y_k, y_{k-1})$. Pomerance's early-abort analysis in [45] says, in some cases, that $\#S_k$ is not much larger than this. We heuristically assume that this is true in more generality.

The approximation $\Psi(x_i, y_i, y_{i-1}) \approx x_i/u_i^{u_i}$, where $u_i = (\log x_i)/\log y_i$, now implies that $\#S_k$ is approximately $x/(u_1^{u_1} \cdots u_k^{u_k})$. More generally, $\#S_i$ is approximately $x/(u_1^{u_1} \cdots u_i^{u_i})$.

Write T_i for the cost of finding the y_i-smooth part of an integer. The early-abort factorization method, applied to a uniform random element of S_0, always takes time T_1 to find primes $\le y_1$; with probability $\#S_1/\#S_0 \approx 1/u_1^{u_1}$ takes additional time T_2 to find primes $\le y_2$; with probability $\#S_2/\#S_0 \approx 1/(u_1^{u_1} u_2^{u_2})$ takes additional time T_3 to find primes $\le y_3$; and so on. With probability $\#S_k/\#S_0 \approx 1/(u_1^{u_1} \cdots u_k^{u_k})$ an integer is y-smooth and survives all aborts.

Balancing the time for the early-abort stages, i.e., ensuring that each stage takes time approximately T_1, requires choosing x_1 (depending on y_1) so that $u_1^{u_1} \approx T_2/T_1$, choosing x_2 (depending on y_2) so that $u_2^{u_2} \approx T_3/T_2$, and so on through choosing x_{k-1} (depending on y_{k-1}) so that $u_{k-1}^{u_{k-1}} \approx T_k/T_{k-1}$. Then x_k is determined as $x/(x_1 \cdots x_{k-1})$, and u_k is determined as $(\log x_k)/\log y_k = u - (\log x_1 \cdots x_{k-1})/\log y = u - (\theta_1 u_1 + \theta_2 u_2 + \cdots + \theta_{k-1} u_{k-1})$ where $\theta_i = (\log y_i)/\log y$.

As a special case (including the cases considered by Pomerance), if all u_i are in $u^{1+o(1)}$, then $T_{i+1}/T_i \approx u_i^{u_i}$ is a $1 + o(1)$ power of u^{u_i}, so $u_1^{u_1} \cdots u_k^{u_k}$ is a $1 + o(1)$ power of $u^{u_1 + \cdots + u_k} = u^{u + u_1(1-\theta_1) + \cdots + u_{k-1}(1-\theta_{k-1})}$, which is a $1 + o(1)$ power of

$$\begin{aligned} &u^u (T_2/T_1)^{1-\theta_1} \cdots (T_k/T_{k-1})^{1-\theta_{k-1}} \\ &\quad = u^u T_1^{\theta_1} T_2^{\theta_2 - \theta_1} T_3^{\theta_3 - \theta_2} \cdots T_{k-1}^{\theta_{k-1} - \theta_{k-2}} T_k^{1-\theta_{k-1}}/T_1. \end{aligned}$$

In other words, compared to the original smoothness probability $1/u^u$ of integers in S_0, the found-by-early-abort-factorization probability is smaller by a factor

$T_1^{\theta_1} T_2^{\theta_2-\theta_1} \cdots T_{k-1}^{\theta_{k-1}-\theta_{k-2}} T_k^{1-\theta_{k-1}}/T_1$. The time for all stages of early-abort factorization is essentially T_1. For example, for $\theta_i = i/k$, the product of the time and the loss factor is $(T_1 T_2 \cdots T_k)^{1/k}$.

We see two obstacles to proving the formula $(T_1 T_2 \cdots T_k)^{1/k}$ for early-abort ECM. First, the assumption $u_i \in u^{1+o(1)}$ is correct for exponential-time smoothness tests for standard ranges of x and y; but $u_i \in u^{0.5+o(1)}$ for ECM, except for $i = k$. Second, the error factor $u^{o(u)}$ in the standard u^u approximation is larger than the entire ECM running time. Despite these caveats we conjecture that the heuristics apply beyond the case of exponential-time smoothness tests, and in particular apply to early-abort ECM.

Even when smoothness theorems are available, one should not overstate the extent to which they constitute rigorous analyses of NFS. There is no proof that NFS congruences have similar smoothness probability to uniform random integers; this is one of the NFS heuristics. There is no proof that ECM finds all small primes at similar speed; this is another heuristic. As mentioned earlier, Pomerance's analysis in [45] actually uses the provable Pollard–Strassen smoothness-detection method, and Bernstein's batch trial-division method [8] is proven to run in polynomial time per input; but both of these methods perform poorly in the *AT* metric. Similarly, Pomerance proved in [45] that Dixon's random squares have similar smoothness probability to uniform random integers; but Dixon's method is much slower than NFS, and proving something similar about NFS is an open problem.

3.5. Impact of early aborts on smoothness probabilities. Because early-abort ECM does not find *all* smooth values, it forces batch NFS to consider more pairs (a, b), and therefore slightly larger pairs (a, b). This increase means that the auxiliary integers c are larger and less likely to be smooth. We conclude by showing that this effect does not eliminate the (heuristic) asymptotic gain produced by early aborts.

Recall that the smoothness probability of c is heuristically $1/v^v$, where v is the ratio of the number of bits in $(|f_d| + \cdots + |f_0|)H^d$ and the number of bits in y. The derivative of v with respect to $\log H$ is $d/\log y$, so the derivative of $\log(v^v)$ with respect to $\log H$ is $d(1 + \log v)/\log y \in 1/(3\gamma\mu) + o(1)$; here we have used the asymptotics $d \in (1/\mu + o(1))(\log N)^{1/3}(\log\log N)^{-1/3}$, $\log y \in (\gamma + o(1))(\log N)^{1/3}(\log\log N)^{2/3}$, and $\log v \in (1/3 + o(1))\log\log N$.

Write $\delta = (2/3)/(2 - 1/(3\gamma\mu))$. Multiplying H by a factor $T^{\delta+o(1)}$ means multiplying the number of pairs (a, b) by a factor $T^{2\delta+o(1)}$ and thus multiplying the number of smoothness tests by a factor $T^{2\delta+o(1)}$. Meanwhile it multiplies v^v by a factor $T^{\delta/(3\gamma\mu)+o(1)}$, and thus multiplies the final number of smooth congruences by a factor $T^{(2-1/(3\gamma\mu))\delta+o(1)} = T^{2/3+o(1)}$. Our heuristics state that switching from ECM to early-abort ECM reduces the number of smooth congruences found by a factor $T^{2/3+o(1)}$, producing just enough smooth congruences for a successful factorization, while decreasing the cost of each smoothness test by a factor $T^{1+o(1)}$. The overall speedup factor is $T^{1-2\delta+o(1)}$.

For example, [7] took $\gamma \approx 0.790420$ and $\mu \approx 0.702860$, so the speedup factor is $T^{0.047\ldots+o(1)}$. As another example, batch NFS with $\beta = 0.5$ takes $\gamma \approx 0.681600$ and $\mu \approx 0.921150$, so the speedup factor is $T^{0.092\ldots+o(1)}$.

References

[1] — (no editor), *Proceedings of the 18th annual ACM symposium on theory of computing*, Association for Computing Machinery, 1986. ISBN 0-89791-193-8. See [50].

[2] Michel Abdalla, Paulo S. L. M. Barreto (editors), *Progress in cryptology — LATINCRYPT 2010, first international conference on cryptology and information security in Latin America, Puebla, Mexico, August 8–11, 2010, proceedings*, Lecture Notes in Computer Science, 6212, Springer, 2010. See [9].

[3] A. O. L. Atkin, Francois Morain, *Finding suitable curves for the elliptic curve method of factorization*, Mathematics of Computation **60** (1993), 399–405. ISSN 0025–5718. MR 93k:11115. URL: http://www.lix.polytechnique.fr/~morain/Articles/articles.english.html. Citations in this document: §A.3.

[4] Shi Bai, Cyril Bouvier, Alain Filbois, Pierrick Gaudry, Laurent Imbert, Alexander Kruppa, Francois Morain, Emmanuel Thomé, Paul Zimmermann, *CADO-NFS — Crible Algébrique: Distribution, Optimisation — Number Field Sieve* (2013). URL: http://cado-nfs.gforge.inria.fr/. Citations in this document: §1.

[5] Razvan Barbulescu, Joppe W. Bos, Cyril Bouvier, Thorsten Kleinjung, Peter L. Montgomery, *Finding ECM-friendly curves through a study of Galois properties* (2013), 63–86, *ANTS-X: proceedings of the tenth Algorithmic Number Theory Symposium*, 2013. URL: http://msp.org/obs/2013/1/p04.xhtml. Citations in this document: §A.3.

[6] Elaine Barker, William Barker, William Burr, William Polk, Miles Smid, *Recommendation for key management — part 1: general (revision 3)* (2012). URL: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf. Citations in this document: §1.2.

[7] Daniel J. Bernstein, *Circuits for integer factorization: a proposal* (2001). URL: http://cr.yp.to/papers.html#nfscircuit. Citations in this document: §1.1, §1.3, §1.3, §1.3, §1.3, §2.4, §2.5, §2.11, §3.5.

[8] Daniel J. Bernstein, *How to find small factors of integers* (2002). URL: http://cr.yp.to/papers.html#sf. Citations in this document: §3.4.

[9] Daniel J. Bernstein, Peter Birkner, Tanja Lange, *Starfish on strike*, in Latincrypt 2010 [2] (2010), 61–80. URL: http://eprint.iacr.org/2010/367. Citations in this document: §A, §A.3.

[10] Daniel J. Bernstein, Peter Birkner, Tanja Lange, Christiane Peters, *ECM using Edwards curves*, Mathematics of Computation **82** (2013), 1139–1179. URL: http://eprint.iacr.org/2008/016. Citations in this document: §A, §A.3, §A.3.

[11] Daniel J. Bernstein, Hsueh-Chung Chen, Ming-Shing Chen, Chen-Mou Cheng, Chun-Hung Hsiao, Tanja Lange, Zong-Cing Lin, Bo-Yin Yang, *The billion-mulmod-per-second PC*, in Workshop record of SHARCS'09: special-purpose hardware for attacking cryptographic systems (2009), 131–144. URL: http://www.hyperelliptic.org/tanja/SHARCS/record2.pdf. Citations in this document: §1.2.

[12] Daniel J. Bernstein, Tien-Ren Chen, Chen-Mou Cheng, Tanja Lange, Bo-Yin Yang, *ECM on graphics cards*, in Eurocrypt 2009 [28] (2009), 483–501. URL: http://eprint.iacr.org/2008/480/. Citations in this document: §1.2.

[13] Daniel J. Bernstein, Tanja Lange, *Faster addition and doubling on elliptic curves*, in Asiacrypt 2007 [**33**] (2007), 29–50. URL: `http://cr.yp.to/newelliptic/`. Citations in this document: §A.1.

[14] Daniel J. Bernstein, Tanja Lange, *Non-uniform cracks in the concrete: the power of free precomputation*, in Asiacrypt 2013 [**49**] (2013), 321–340. URL: `http://eprint.iacr.org/2012/318`. Citations in this document: §1.3, §1.3, §1.3, §2.5.

[15] Gianfranco Bilardi, Franco P. Preparata, *Horizons of parallel computation*, Journal of Parallel and Distributed Computing **27** (1995), 172–182. Citations in this document: §1.1.

[16] Dan Boneh (editor), *Advances in cryptology — CRYPTO 2003, 23rd annual international cryptology conference, Santa Barbara, California, USA, August 17–21, 2003*, Lecture Notes in Computer Science, 2729, Springer, 2003. ISBN 3-540-40674-3. See [51].

[17] Joppe W. Bos, Thorsten Kleinjung, *ECM at work*, in Asiacrypt 2012 [**55**] (2012), 467–484. URL: `http://eprint.iacr.org/2012/089`. Citations in this document: §1.2, §A.

[18] Richard P. Brent, H. T. Kung, *The area-time complexity of binary multiplication*, Journal of the ACM **28** (1981), 521–534. URL: `http://wwwmaths.anu.edu.au/~brent/pub/pub055.html`. Citations in this document: §1.1, §1.3.

[19] Joe P. Buhler, Hendrik W. Lenstra, Jr., Carl Pomerance, *Factoring integers with the number field sieve*, in [**36**] (1993), 50–94. Citations in this document: §1.3, §1.3, §1.3, §1.3, §1.3, §1.3, §2.2, §2.3.

[20] Don Coppersmith, *Modifications to the number field sieve*, Journal of Cryptology **6** (1993), 169–180. Citations in this document: §1.3, §1.3, §1.3, §1.3, §1.3, §1.3, §1.3, §1.3, §2.3, §2.5, §2.5, §2.11.

[21] Richard Crandall, Carl Pomerance, *Prime numbers. A computational perspective*, Springer-Verlag, New York, 2001. ISBN 0–387–94777–9. MR 2002a:11007. Citations in this document: §3.

[22] Harold M. Edwards, *A normal form for elliptic curves*, Bulletin of the American Mathematical Society **44** (2007), 393–422. URL: `http://www.ams.org/bull/2007-44-03/S0273-0979-07-01153-6/home.html`. Citations in this document: §A.1.

[23] Jens Franke, Thorsten Kleinjung, Christof Paar, Jan Pelzl, Christine Priplata, Colin Stahlke, *SHARK: a realizable special hardware sieving device for factoring 1024-bit integers*, in CHES 2005 [**47**] (2005), 119–130. Citations in this document: §1, §1.1.

[24] Willi Geiselmann, Adi Shamir, Rainer Steinwandt, Eran Tromer, *Scalable hardware for sparse systems of linear equations, with applications to integer factorization*, in CHES 2005 [**47**] (2005), 131–146. Citations in this document: §1, §1.1.

[25] Nadia Heninger, Zakir Durumeric, Eric Wustrow, J. Alex Halderman, *Mining your Ps and Qs: detection of widespread weak keys in network devices*, in Proceedings of the 21st USENIX Security Symposium (2012). URL: `https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger`. Citations in this document: §1.1.

[26] Florian Hess, Sebastian Pauli, Michael E. Pohst (editors), *Algorithmic number theory, 7th international symposium, ANTS-VII, Berlin, Germany, July 23–28, 2006*, Lecture Notes in Computer Science, 4076, Springer, Berlin, 2006. ISBN 3-540-36075-1. See [57].

[27] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, Ed Dawson, *Twisted Edwards curves revisited*, in Asiacrypt 2008 [**44**] (2008), 326–242. URL: `http://eprint.iacr.org/2008/552`. Citations in this document: §A.1.

[28] Antoine Joux (editor), *Advances in cryptology — Eurocrypt 2009, 28th annual international conference on the theory and applications of cryptographic techniques, Cologne, Germany, April 26–30, 2009*, Lecture Notes in Computer Science, 5479, Springer, 2009. See [12].

[29] Thorsten Kleinjung, *On polynomial selection for the general number field sieve*, Mathematics of Computation **75** (2006), 2037–2047. Citations in this document: §1.

[30] Thorsten Kleinjung, *Polynomial selection. Slides presented at the CADO workshop, Nancy, France* (2008). URL: http://cado.gforge.inria.fr/workshop/slides/kleinjung.pdf. Citations in this document: §1.

[31] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev, Paul Zimmermann, *Factorization of a 768-bit RSA modulus*, in Crypto 2010 [**46**] (2010), 333–350. Citations in this document: §1.

[32] Olaf M. Kolkman, Matthijs Mekking, Miek Gieben, *RFC 6781: DNSSEC operational practices, version 2* (2012). URL: http://tools.ietf.org/html/rfc6781. Citations in this document: §1.

[33] Kaoru Kurosawa (editor), *Advances in cryptology — ASIACRYPT 2007, 13th international conference on the theory and application of cryptology and information security, Kuching, Malaysia, December 2–6, 2007*, Lecture Notes in Computer Science, 4833, Springer, 2007. See [13].

[34] Chi-Sung Laih (editor), *Advances in cryptology — ASIACRYPT 2003, 9th international conference on the theory and application of cryptology and information security, Taipei, Taiwan, November 30–December 4, 2003*, Lecture Notes in Computer Science, 2894, Springer, 2003. ISBN 3-540-20592-6. See [38].

[35] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, Christophe Wachter, *Public keys*, in Crypto 2012 [**48**] (2012), 626–642. Citations in this document: §1.1.

[36] Arjen K. Lenstra, Hendrik W. Lenstra, Jr. (editors), *The development of the number field sieve*, Lecture Notes in Mathematics, 1554, Springer, 1993. ISBN 3-540-57013-6. MR 96m:11116. Citations in this document: §1, §2.2. See [19].

[37] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., Mark S. Manasse, John M. Pollard, *The factorization of the ninth Fermat number*, Mathematics of Computation **61** (1993), 319–349. ISSN 0025–5718. MR 93k:11116. Citations in this document: §1.3.

[38] Arjen K. Lenstra, Eran Tromer, Adi Shamir, Wil Kortsmit, Bruce Dodson, James Hughes, Paul C. Leyland, *Factoring estimates for a 1024-bit RSA modulus*, in Asiacrypt 2003 [**34**] (2003), 55–74. Citations in this document: §1.

[39] Hendrik W. Lenstra, Jr., *Factoring integers with elliptic curves*, Annals of Mathematics **126** (1987), 649–673. ISSN 0003–486X. MR 89g:11125. URL: http://links.jstor.org/sici?sici=0003-486X(198711)2:126:3<649:FIWEC>2.0.CO;2-V. Citations in this document: §A.

[40] Hendrik W. Lenstra, Jr., Robert Tijdeman (editors), *Computational methods in number theory I*, Mathematical Centre Tracts, 154, Mathematisch Centrum, Amsterdam, 1982. ISBN 90–6196–248–X. MR 84c:10002. See [45].

[41] Edward Lewis, *DNSSEC at TLDs, start of 4Q 2013* (2013). URL: https://elists.isoc.org/pipermail/dnssec-coord/2013-October/000172.html. Citations in this document: §1.

[42] James McKee, *Subtleties in the distribution of the numbers of points on elliptic curves over a finite prime field*, Journal of the London Mathematical Society **59** (1999), 448–460. URL: http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=20335. Citations in this document: §A.3.

[43] Michael A. Morrison, John Brillhart, *A method of factoring and the factorization of F_7*, Mathematics of Computation **29** (1975), 183–205. ISSN 0025–5718. MR 51:8017. Citations in this document: §3.1.

[44] Josef Pieprzyk (editor), *Advances in cryptology — ASIACRYPT 2008, 14th international conference on the theory and application of cryptology and information security, Melbourne, Australia, December 7–11, 2008*, Lecture Notes in Computer Science, 5350, Springer, 2008. See [27].

[45] Carl Pomerance, *Analysis and comparison of some integer factoring algorithms*, in [**40**] (1982), 89–139. MR 84i:10005. URL: http://cr.yp.to/bib/1982/pomerance.html. Citations in this document: §3.1, §3.4, §3.4, §3.4, §3.4, §3.4.

[46] Tal Rabin (editor), *Advances in cryptology — CRYPTO 2010, 30th annual cryptology conference, Santa Barbara, CA, USA, August 15–19, 2010*, Lecture Notes in Computer Science, 6223, Springer, 2010. ISBN 978-3-642-14622-0. See [31].

[47] Josyula R. Rao, Berk Sunar (editors), *Cryptographic hardware and embedded systems — CHES 2005, 7th international workshop, Edinburgh, UK, August 29–September 1, 2005*, Lecture Notes in Computer Science, 3659, Springer, 2005. ISBN 3-540-28474-5. See [23], [24].

[48] Reihaneh Safavi-Naini, Ran Canetti (editors), *Advances in cryptology — CRYPTO 2012, 32nd annual cryptology conference, Santa Barbara, CA, USA, August 19–23, 2012*, Lecture Notes in Computer Science, 7417, Springer, 2012. ISBN 978-3-642-32008-8. See [35].

[49] Kazue Sako, Palash Sarkar (editors), *Advances in cryptology — Asiacrypt 2013 — 19th international conference on the theory and application of cryptology and information security, Bengaluru, India, December 1–5, 2013, proceedings, part II*, Lecture Notes in Computer Science, 8270, Springer, 2013. ISBN 978-3-642-42045-0. See [14].

[50] Claus P. Schnorr, Adi Shamir, *An optimal sorting algorithm for mesh-connected computers*, in [**1**] (1986), 255–261. Citations in this document: §1.1, §2.6.

[51] Adi Shamir, Eran Tromer, *Factoring large numbers with the TWIRL device*, in Crypto 2003 [**16**] (2003), 1–26. Citations in this document: §1, §1.1, §1.2.

[52] Joseph H. Silverman, *The arithmetic of elliptic curves*, Graduate Texts in Mathematics, 106, Springer, New York, 1986. Citations in this document: §A.3.

[53] C. D. Thompson, H. T. Kung, *Sorting on a mesh-connected parallel computer*, Communications of the ACM **20** (1977), 263–271. ISSN 0001-0782. Citations in this document: §1.1.

[54] Paul C. van Oorschot, Michael Wiener, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12** (1999), 1–28. ISSN 0933-2790. URL: http://members.rogers.com/paulv/papers/pubs.html. Citations in this document: §1.1.

[55] Xiaoyun Wang, Kazue Sako (editors), *Advances in cryptology — ASIACRYPT 2012, 18th international conference on the theory and application of cryptology and information security, Beijing, China, December 2–6, 2012, proceedings*, Lecture Notes in Computer Science, 7658, Springer, 2012. ISBN 978-3-642-34960-7. See [17].

[56] Michael J. Wiener, *The full cost of cryptanalytic attacks*, Journal of Cryptology **17** (2004), 105–124. ISSN 0933–2790. URL: http://sites.google.com/site/michaeljameswiener/. Citations in this document: §1.1.

[57] Paul Zimmermann, Bruce Dodson, *20 years of ECM*, in ANTS 2006 [**26**] (2006), 525–542. Citations in this document: §A.

A ECM

The elliptic-curve method of factorization (ECM) was introduced by H. Lenstra in 1987 [**39**]; [**57**] gives a good overview of ECM and a description of the most popular ECM software, GMP-ECM. The use of Edwards curves in ECM was suggested by Bernstein, Birkner, Lange, and Peters in [**10**] and implemented for amd64 architectures; see http://eecm.cr.yp.to/mpfq.html. A fast implementation on GPUs of the particularly efficient curves of [**9**] was presented at Asiacrypt 2012 by Bos and Kleinjung in [**17**]. This appendix gives a brief overview of Edwards curves and ECM.

A.1. Edwards curves. An Edwards curve [**22**] over a field with $2 \neq 0$ is given by an equation of the form $x^2+y^2 = 1+dx^2y^2$, for some $d \notin \{0,1\}$. The addition law on an Edwards curve is given by

$$(x_1,y_1) \oplus (x_2,y_2) = \left(\frac{x_1y_2+y_1x_2}{1+dx_1x_2y_1y_2}, \frac{y_1y_2-x_1x_2}{1-dx_1x_2y_1y_2}\right).$$

The neutral element is $(0,1)$; $(0,-1)$ has order 2; $(\pm 1,0)$ have order 4.

We presented projective addition formulas for Edwards curves in [**13**]. Subsequently several papers generalized the curve shape to twisted Edwards curves and improved the addition laws; the most efficient arithmetic is due to Hisil, Wong, Carter, and Dawson in [**27**]. For an overview of the costs of elliptic-curve arithmetic in various representations see http://hyperelliptic.org/EFD/.

A.2. The elliptic-curve method of factorization. ECM is a variant of the $p-1$ method: it uses elliptic curves modulo c instead of the multiplicative group modulo c, where c is the number to be factored.

The $p-1$ method picks a random positive integer $a < c$ and a number s and computes $\gcd\{a^s-1,c\}$. In the $p-1$ method a factor p of c is found, meaning that p divides the gcd, if the order of a modulo p is a divisor of s. To make this very likely for many factors of c, one chooses s as a very smooth number: typically $s = \operatorname{lcm}\{1,2,3,\ldots,B_1\}$ for some smoothness bound B_1.

ECM replaces a with a point P on an elliptic curve modulo c, and replaces exponentiation with scalar multiplication by s: it computes $R = [s]P$. We describe ECM with the elliptic curve instantiated as an Edwards curve. The computations on the curve modulo c use the same addition law as over the rational numbers $\mathbf{Q}$ where we reduce all results modulo c.

Let again p be a factor of c and assume that the discriminant of the curve and all of the denominators in the coefficients of E and P are coprime with c (otherwise factors of c would already be found). Define P_p and R_p as the reductions of the points P and R respectively modulo p. If the order of P_p is a divisor of s then $R_p = (0,1)$, so $\gcd\{x(R), c\}$ is divisible by p, where $x(R)$ denotes the x-coordinate of R. This part of the computation is usually called stage 1 of ECM. Stage 2 pushes the limit of the largest prime in $\operatorname{ord}(P_p)$ that can be found with the method; a simple form of stage 2 computes $R_1 = [p_{k+1}]R, R_2 = [p_{k+2}]R, \ldots, R_\ell = [p_{k+\ell}]R$, where $p_{k+1}, p_{k+2}, \ldots, p_{k+\ell}$ are the primes between B_1 and another bound B_2, followed by computing $\gcd\{x(R_1)x(R_2)\cdots x(R_\ell), c\}$.

Efficient implementations use windowing methods with precomputations to compute $[s]P$ and compute the R_i in succession by first computing the differences between successive values $[p_{i+1} - p_i]R$ for $i \in \{k+1, \ldots, k+\ell-1\}$, namely $[2]R$, $[4]R, \ldots$

The main advantage of ECM over the $p-1$ method is that the curve can be varied; per number c many curves can be tried; this increases the probability that the order of one of the points P is smooth modulo one of the factors p. The $p-1$ method is limited to the one group of order $p-1$; if $p-1$ is not smooth for any factor of c this method will not succeed while it is impossible to have primes p so that the order of any elliptic curve modulo p has only large factors.

A.3. Choice of elliptic curve. The choice of elliptic curve is influenced by the efficiency of scalar multiplication but also by the likelihood of leading to a factorization. The factor p is found if $\operatorname{ord}(P_p)$ is a factor of s or of $s \cdot p_i$ for some $i \in \{k+1, k+2, \ldots, k+\ell\}$. The order of P_p divides the order of E which is in the Hasse interval around p, namely $[p+1-2\sqrt{p}, p+1+2\sqrt{p}]$. McKee [42] showed that orders of elliptic curves are more likely to be smooth than general numbers of this size. Orders of Edwards curves have even larger smoothness probability because they have a guaranteed cofactor of 4 in the group order. For more details on smoothness chances see [10], [9], and [5].

It is possible to further increase the smoothness chances by choosing curves that have a large $\mathbf{Q}$-rational torsion subgroup, i.e., for which the curve over $\mathbf{Q}$ has many points of finite order. Mazur's theorem (see, e.g., [52]) limits the maximal number of points of finite order to 16. Atkin and Morain [3] gave a construction of curves attaining this maximum number of points of finite order which was translated to the setting of Edwards curves in [10]. This construction provides a family of curves E and points P, where it is ensured that P is none of the points of finite order (otherwise $\gcd\{x(R), c\}$ would be divisible by c itself).