Factoring Large Numbers with the TWIRL Device

Adi Shamir and Eran Tromer

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot 76100, Israel
{shamir,tromer}@wisdom.weizmann.ac.il

Abstract. The security of the RSA cryptosystem depends on the difficulty of factoring large integers. The best current factoring algorithm is the Number Field Sieve (NFS), and its most difficult part is the sieving step. In 1999 a large distributed computation involving hundreds of workstations working for many months managed to factor a 512-bit RSA key, but 1024-bit keys were believed to be safe for the next 15-20 years. In this paper we describe a new hardware implementation of the NFS sieving step (based on standard 0.13μm, 1GHz silicon VLSI technology) which is 3-4 orders of magnitude more cost effective than the best previously published designs (such as the optoelectronic TWINKLE and the mesh-based sieving). Based on a detailed analysis of all the critical components (but without an actual implementation), we believe that the NFS sieving step for 512-bit RSA keys can be completed in less than ten minutes by a $10K device. For 1024-bit RSA keys, analysis of the NFS parameters (backed by experimental data where possible) suggests that sieving step can be completed in less than a year by a $10M device. Coupled with recent results about the cost of the NFS matrix step, this raises some concerns about the security of this key size.

1 Introduction

The hardness of integer factorization is a central cryptographic assumption and forms the basis of several widely deployed cryptosystems. The best integer factorization algorithm known is the Number Field Sieve [12], which was successfully used to factor 512-bit and 530-bit RSA moduli [5, 1]. However, it appears that a PC-based implementation of the NFS cannot practically scale much further, and specifically its cost for 1024-bit composites is prohibitive. Recently, the prospect of using custom hardware for the computationally expensive steps of the Number Field Sieve has gained much attention. While mesh-based circuits for the matrix step have rendered that step quite feasible for 1024-bit composites [3, 16], the situation is less clear concerning the sieving step. Several sieving devices have been proposed, including TWINKLE [19, 15] and a mesh-based circuit [7], but apparently none of these can practically handle 1024-bit composites.

One lesson learned from Bernstein's mesh-based circuit for the matrix step [3] is that it is inefficient to have memory cells that are "simply sitting around, twiddling their thumbs" — if merely storing the input is expensive, we should utilize it efficiently by appropriate parallelization. We propose a new device that combines this intuition with the TWINKLE-like approach of exchanging time and space. Whereas TWINKLE tests sieve location one by one serially, the new device handles thousands of sieve locations in parallel at every clock cycle. In addition, it is smaller and easier to construct: for 512-bit composites we can fit 79 independent sieving devices on a 30cm single silicon wafer, whereas each TWINKLE device requires a full GaAs wafer. While our approach is related to [7], it scales better and avoids some thorny issues.

The main difficulty is how to use a single copy of the input (or a small number of copies) to solve many subproblems in parallel, without collisions or long propagation delays and while maintaining storage efficiency. We address this with a heterogeneous design that uses a variety of routing circuits and takes advantage of available technological tradeoffs. The resulting cost estimates suggest that for 1024-bit composites the sieving step may be surprisingly feasible.

Section 2 reviews the sieving problem and the TWINKLE device. Section 3 describes the new device, called TWIRL[1], and Section 4 provides preliminary cost estimates. Appendix A discusses additional design details and improvements. Appendix B specifies the assumptions used for the cost estimates, and Appendix C relates this work to previous ones.

2 Context

2.1 Sieving in the Number Field Sieve

Our proposed device implements the sieving substep of the NFS relation collection step, which in practice is the most expensive part of the NFS algorithm [16]. We begin by reviewing the sieving problem, in a greatly simplified form and after appropriate reductions.[2] See [12] for background on the Number Field Sieve.

The inputs of the sieving problem are $R \in \mathbb{Z}$ (*sieve line width*), $T > 0$ (*threshold*) and a set of pairs (p_i,r_i) where the p_i are the prime numbers smaller than some *factor base bound* B. There is, on average, one pair per such prime. Each pair (p_i,r_i) corresponds to an arithmetic progression $P_i = \{a : a \equiv r_i \pmod{p_i}\}$. We are interested in identifying the sieve locations $a \in \{0,\ldots,R-1\}$ that are members of many progressions P_i with large p_i:

$$g(a) > T \quad \text{where} \quad g(a) = \sum_{i:a \in P_i} \log_h p_i$$

for some fixed h (possibly $h > 2$). It is permissible to have "small" errors in this threshold check; in particular, we round all logarithms to the nearest integer.

In the NFS relation collection step we have two types of sieves: *rational* and *algebraic*. Both are of the above form, but differ in their factor base bounds (B_R

[1] TWIRL stands for "The Weizmann Institute Relation Locator"

[2] The description matches both line sieving and lattice sieving. However, for lattice sieving we may wish to take a slightly different approach (cf. A.8).

vs. B_{A}), threshold T and basis of logarithm h. We need to handle H *sieve lines*, and for sieve line both sieves are performed, so there are $2H$ sieving instances overall. For each sieve line, each value a that passes the threshold in both sieves implies a *candidate*. Each candidate undergoes additional tests, for which it is beneficial to also know the set $\{i : a \in P_i\}$ (for each sieve separately). The most expensive part of these tests is *cofactor factorization*, which involves factoring medium-sized integers.[3] The candidates that pass the tests are called *relations*. The output of the relation collection step is the list of relations and their corresponding $\{i : a \in P_i\}$ sets. Our goal is to find a certain number of relations, and the parameters are chosen accordingly a priori.

2.2 TWINKLE

Since TWIRL follows the TWINKLE [19, 15] approach of exchanging time and space compared to traditional NFS implementations, we briefly review TWINKLE (with considerable simplification). A TWINKLE device consists of a wafer containing numerous independent cells, each in charge of a single progression P_i. After initialization the device operates for R clock cycles, corresponding to the sieving range $\{0 \leq a < R\}$. At clock cycle a, the cell in charge of the progression P_i emits the value $\log p_i$ iff $a \in P_i$. The values emitted at each clock cycle are summed, and if this sum exceeds the threshold T then the integer a is reported. This event is announced back to the cells, so that the i values of the pertaining P_i is also reported. The global summation is done using analog optics; clocking and feedback are done using digital optics; the rest is implemented by digital electronics. To support the optoelectronic operations, TWINKLE uses Gallium Arsenide wafers which are small, expensive and hard to manufacture compared to silicon wafers, which are readily available.

3 The New Device

3.1 Approach

We next describe the TWIRL device. The description in this section applies to the rational sieve; some changes will be made for the algebraic sieve (cf. A.6), since it needs to consider only a values that passed the rational sieve.

For the sake of concreteness we provide numerical examples for a plausible choice of parameters for 1024-bit composites.[4] This choice will be discussed in Sections 4 and B.2; it is not claimed to be optimal, and all costs should be taken as rough estimates. The concrete figures will be enclosed in double angular brackets: $\langle\!\langle x \rangle\!\rangle_{\mathrm{R}}$ and $\langle\!\langle x \rangle\!\rangle_{\mathrm{A}}$ indicate values for the algebraic and rational sieves respectively, and $\langle\!\langle x \rangle\!\rangle$ is applicable to both.

We wish to solve $H \,\langle\!\langle \approx 2.7 \cdot 10^8 \rangle\!\rangle$ pairs of instances of the sieving problem, each of which has sieving line width $R \,\langle\!\langle = 1.1 \cdot 10^{15} \rangle\!\rangle$ and smoothness bound

[3] We assume use of the "2+2 large primes" variant of the NFS [12, 13].

[4] This choice differs considerably from that used in preliminary drafts of this paper.

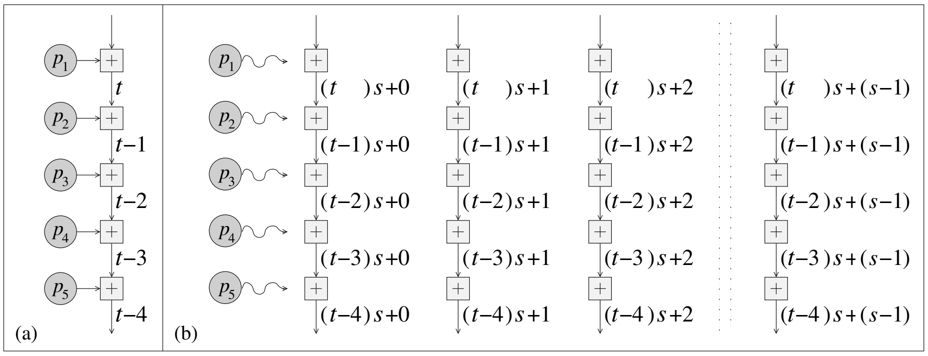

Fig. 1: Flow of sieve locations through the device in (a) a chain of adders and (b) TWIRL.

$B \langle\!\langle = 3.5 \cdot 10^9 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle = 2.6 \cdot 10^{10} \rangle\!\rangle_{\mathrm{A}}$. Consider first a device that handles one sieve location per clock cycle, like TWINKLE, but does so using a pipelined systolic chain of electronic adders.[5] Such a device would consist of a long unidirectional bus, $\log_2 T \langle\!\langle = 10 \rangle\!\rangle$ bits wide, that connects millions of conditional adders in series. Each conditional adder is in charge of one progression P_i; when activated by an associated timer, it adds the value[6] $\lfloor \log p_i \rceil$ to the bus. At time t, the z-th adder handles sieve location $t - z$. The first value to appear at the end of the pipeline is $g(0)$, followed by $g(1), \ldots, g(R)$, one per clock cycle. See Fig. 1(a).

We reduce the run time by a factor of $s \langle\!\langle = 4{,}096 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle = 32{,}768 \rangle\!\rangle_{\mathrm{A}}$ by handling the sieving range $\{0, \ldots, R-1\}$ in chunks of length s, as follows. The bus is thickened by a factor of s to contain s logical lines of $\log_2 T$ bits each. As a first approximation (which will be altered later), we may think of it as follows: at time t, the z-th stage of the pipeline handles the sieve locations $(t-z)s + i$, $i \in \{0, \ldots, s-1\}$. The first values to appear at the end of the pipeline are $\{g(0), \ldots, g(s-1)\}$; they appear simultaneously, followed by successive disjoint groups of size s, one group per clock cycle. See Fig. 1(b).

Two main difficulties arise: the hardware has to work s times harder since time is compressed by a factor of s, and the additions of $\lfloor \log p_i \rceil$ corresponding to the same given progression P_i can occur at different lines of a thick pipeline. Our goal is to achieve this parallelism without simply duplicating all the counters and adders s times. We thus replace the simple TWINKLE-like cells by other units which we call *stations*. Each station handles a small portion of the progressions, and its interface consists of bus input, bus output, clock and some circuitry for loading the inputs. The stations are connected serially in a pipeline, and at the end of the bus (i.e., at the output of the last station) we place a threshold check unit that produces the device output.

An important observation is that the progressions have periods p_i in a very large range of sizes, and different sizes involve very different design tradeoffs. We

[5] This variant was considered in [15], but deemed inferior in that context.

[6] $\lfloor \log p_i \rceil$ denote the value $\log_h p_i$ for some fixed h, rounded to the nearest integer.

thus partition the progressions into three classes according to the size of their p_i values, and use a different station design for each class. In order of decreasing p_i value, the classes will be called *largish*, *smallish* and *tiny*.[7]

This heterogeneous approach leads to reasonable device sizes even for 1024-bit composites, despite the high parallelism: using standard VLSI technology, we can fit $\langle\!\langle 4 \rangle\!\rangle_{\mathrm{R}}$ rational-side TWIRLs into a single 30cm silicon wafer (whose manufacturing cost is about \$5,000 in high volumes; handling local manufacturing defects is discussed in A.9). Algebraic-side TWIRLs use higher parallelism, and we fit $\langle\!\langle 1 \rangle\!\rangle_{\mathrm{A}}$ of them into each wafer.

The following subsections describe the hardware used for each class of progressions. The preliminary cost estimates that appear later are based on a careful analysis of all the critical components of the design, but due to space limitations we omit the descriptions of many finer details. Some additional issues are discussed in Appendix A.

3.2 Largish Primes

Progressions whose p_i values are much larger than s emit $\lfloor \log p_i \rceil$ values very seldom. For these largish primes $\langle\!\langle p_i > 5.2 \cdot 10^5 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle p_i > 4.2 \cdot 10^6 \rangle\!\rangle_{\mathrm{A}}$, it is beneficial to use expensive logic circuitry that handles many progressions but allows very compact storage of each progression. The resultant architecture is shown in Fig. 2. Each progression is represented as a *progression triplet* that is stored in a memory bank, using compact DRAM storage. The progression triplets are periodically inspected and updated by special-purpose processors, which identify emissions that should occur in the "near future" and create corresponding *emission triplets*. The emission triplets are passed into *buffers* that merge the outputs of several processors, perform fine-tuning of the timing and create *delivery pairs*. The delivery pairs are passed to pipelined *delivery lines*, consisting of a chain of *delivery cells* which carry the delivery pairs to the appropriate bus line and add their $\lfloor \log p_i \rceil$ contribution.

Scanning the progressions. The progressions are partitioned into many $\langle\!\langle 8{,}490 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 59{,}400 \rangle\!\rangle_{\mathrm{A}}$ DRAM banks, where each bank contains some d progression $\langle\!\langle 32 \le d < 2.2 \cdot 10^5 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 32 \le d < 2.0 \cdot 10^5 \rangle\!\rangle_{\mathrm{A}}$. A progression P_i is represented by a progression triplet of the form (p_i, ℓ_i, τ_i), where ℓ_i and τ_i characterize the next element $a_i \in P_i$ to be emitted (which is not stored explicitly) as follows. The value $\tau_i = \lfloor a_i/s \rfloor$ is the time when the next emission should be added to the bus, and $\ell_i = a_i \bmod s$ is the number of the corresponding bus line. A processor repeats the following operations, in a pipelined manner:[8]

[7] These are not to be confused with the "large" and "small" primes of the high-level NFS algorithm — all the primes with which we are concerned here are "small" (rather than "large" or in the range of "special-q").

[8] Additional logic related to reporting the sets $\{i : a \in P_i\}$ is described in Appendix A.7.

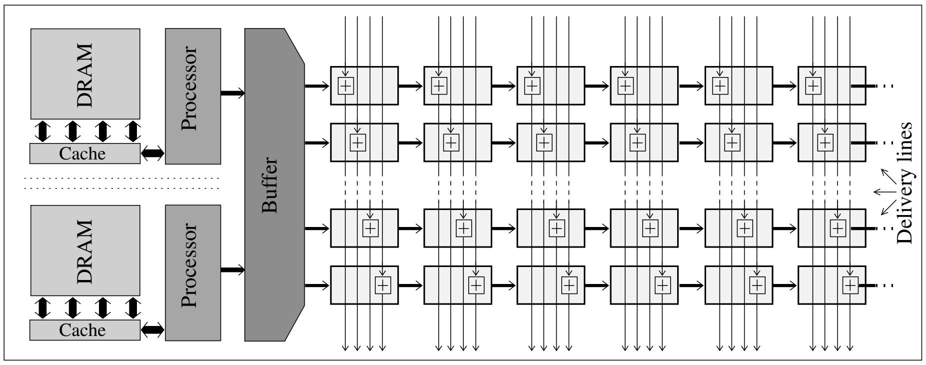

Fig. 2: Schematic structure of a largish station.

1. Read and erase the next state triplet (p_i, ℓ_i, τ_i) from memory.
2. Send an emission triplet $(\lfloor \log p_i \rceil, \ell_i, \tau_i)$ to a buffer connected to the processor.
3. Compute $\ell' \leftarrow (\ell + p) \bmod s$ and $\tau_i' \leftarrow \tau_i + \lfloor p/s \rfloor + w$, where $w = 1$ if $\ell' < \ell$ and $w = 0$ otherwise.
4. Write the triplet (p_i, ℓ_i', τ_i') to memory, according to τ_i' (see below).

We wish the emission triplet $(\lfloor \log p_i \rceil, \ell_i, \tau_i)$ to be created slightly before time τ_i (earlier creation would overload the buffers, while later creation would prevent this emission from being delivered on time). Thus, we need the processor to always read from memory some progression triplet that has an imminent emission. For large d, the simple approach of assigning each emission triplet to a fixed memory address and scanning the memory cyclically would be ineffective. It would be ideal to place the progression triplets in a priority queue indexed by τ_i, but it is not clear how to do so efficiently in a standard DRAM due to its passive nature and high latency. However, by taking advantage of the unique properties of the sieving problem we can get a good approximation, as follows.

Progression storage. The processor reads progression triplets from the memory in sequential cyclic order and at a constant rate ⟪of one triplet every 2 clock cycles⟫. If the value read is empty, the processor does nothing at that iteration. Otherwise, it updates the progression state as above and stores it at a different memory location — namely, one that will be read slightly before time τ_i'. In this way, after a short stabilization period the processor always reads triplets with imminent emissions. In order to have (with high probability) a free memory location within a short distance of any location, we increase the amount of memory ⟪by a factor of 2⟫; the progression is stored at the first unoccupied location, starting at the one that will be read at time τ_i' and going backwards cyclically.

If there is no empty location within ⟪64⟫ locations from the optimal designated address, the progression triplet is stored at an arbitrary location (or a dedicated overflow region) and restored to its proper place at some later stage;

when this happens we may miss a few emissions (depending on the implementation). This happens very seldom,[9] and it is permissible to miss a few candidates.

Autonomous circuitry inside the memory routes the progression triplet to the first unoccupied position preceeding the optimal one. To implement this efficiently we use a two-level memory hierarchy which is rendered possibly by the following observation. Consider a largish processor which is in charge of a set of d adjacent primes $\{p_{\min}, \ldots, p_{\max}\}$. We set the size of the associated memory to $p_{\max}/s$ triplet-sized words, so that triplets with $p_i = p_{\max}$ are stored right before the current read location; triplets with smaller p_i are stored further back, in cyclic order. By the density of primes, $p_{\max} - p_{\min} \approx d \cdot \ln(p_{\max})$. Thus triplet values are always stored at an address that precedes the current read address by at most $d \cdot \ln(p_{\max})/s$, or slightly more due to congestions. Since $\ln(p_{\max}) \leq \ln(B)$ is much smaller than s, memory access always occurs at a small window that slides at a constant rate of one memory location every $\langle\!\langle 2 \rangle\!\rangle$ clock cycles. We may view the $\langle\!\langle 8{,}490 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 59{,}400 \rangle\!\rangle_{\mathrm{A}}$ memory banks as closed rings of various sizes, with an active window "twirling" around each ring at a constant linear velocity.

Each sliding window is handled by a fast SRAM-based cache. Occasionally, the window is shifted by writing the oldest cache block to DRAM and reading the next block from DRAM into the cache. Using an appropriate interface between the SRAM and DRAM banks (namely, read/write of full rows), this hides the high DRAM latency and achieves very high memory bandwidth. Also, this allows simpler and thus smaller DRAM.[10] Note that cache misses cannot occur. The only interface between the processor and memory are the operations "read next memory location" and "write triplet to first unoccupied memory location before the given address". The logic for the latter is implemented within the cache, using auxiliary per-triplet occupancy flags and some local pipelined circuitry.

Buffers. A buffer unit receives emission triplets from several processors in parallel, and sends delivery pairs to several delivery lines. Its task is to convert emission triplets into delivery pairs by merging them where appropriate, fine-tuning their timing and distributing them across the delivery lines: for each received emission triplet of the form $(\lfloor \log p_i \rceil, \ell, \tau)$, the delivery pair $(\lfloor \log p_i \rceil, \ell)$ should be sent to some delivery line (depending on ℓ) at time exactly τ.

Buffer units can be be realized as follows. First, all incoming emission triplets are placed in a parallelized priority queue indexed by τ, implemented as a small

[9] For instance, in simulations for primes close to $\langle\!\langle 20{,}000s \rangle\!\rangle_{\mathrm{R}}$, the distance between the first unoccupied location and the ideal location was smaller than $\langle\!\langle 64 \rangle\!\rangle_{\mathrm{R}}$ for all but $\langle\!\langle 5 \cdot 10^{-6} \rangle\!\rangle_{\mathrm{R}}$ of the iterations. The probability of a random integer $x \in \{1, \ldots, x\}$ having k factors is about $(\log \log x)^{k-1}/(k-1)! \log x$. Since we are (implicitly) sieving over values of size about $x \approx \langle\!\langle 10^{64} \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 10^{101} \rangle\!\rangle_{\mathrm{A}}$ which are "good" (i.e., semi-smooth) with probability $p \approx \langle\!\langle 6.8 \cdot 10^{-5} \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 4.4 \cdot 10^{-9} \rangle\!\rangle_{\mathrm{A}}$, less than $10^{-15}/p$ of the good a's have more than 35 factors; the probability of missing other good a's is negligible.

[10] Most of the peripheral DRAM circuitry (including the refresh circuitry and column decoders) can be eliminated, and the row decoders can be replaced by smaller stateful circuitry. Thus, the DRAM bank can be smaller than standard designs. For the stations that handle the smaller primes in the "largish" range, we may increase the cache size to d and eliminate the DRAM.

mesh whose rows are continuously bubble-sorted and whose columns undergo random local shuffles. The elements in the last few rows are tested for τ matching the current time, and the matching ones are passed to a pipelined network that sorts them by ℓ, merges where needed and passes them to the appropriate delivery lines. Due to congestions some emissions may be late and thus discarded; since the inputs are essentially random, with appropriate choices of parameters this should happen seldom.

The size of the buffer depends on the typical number of time steps that an emission triplet is held until its release time τ (which is fairly small due to the design of the processors), and on the rate at which processors produce emission triplets 《about once per 4 clock cycles》.

Delivery lines. A delivery line receives delivery pairs of the form $(\lfloor \log p_i \rceil, \ell)$ and adds each such pair to bus line ℓ exactly $\lfloor \ell/k \rfloor$ clock cycles after its receipt. It is implemented as a one-dimensional array of cells placed across the bus, where each cell is capable of containing one delivery pair. Here, the j-th cell compares the ℓ value of its delivery pair (if any) to the constant j. In case of equality, it adds $\lfloor \log p_i \rceil$ to the bus line and discards the pair. Otherwise, it passes it to the next cell, as in a shift register.

Overall, there are $《2{,}100120》_{\mathrm{R}}《14{,}900》_{\mathrm{A}}$ delivery lines in the largish stations, and they occupy a significant portion of the device. Appendix A.1 describes the use of interleaved carry-save adders to reduce their cost, and Appendix A.6 nearly eliminates them from the algebraic sieve.

Notes. In the description of the processors, DRAM and buffers, we took the τ values to be arbitrary integers designating clock cycles. Actually, it suffices to maintain these values modulo some integer 《2048》 that upper bounds the number of clock cycles from the time a progression triplet is read from memory to the time when it is evicted from the buffer. Thus, a progression occupies $\log_2 p_i$ + $《\log_2 2048》$ DRAM bits for the triplet, plus $\log_2 p_i$ bits for reinitialization (cf. A.4).

The amortized circuit area per largish progression is $\Theta(s^2(\log s)/p_i + \log s + \log p_i)$.[11] For fixed s this equals $\Theta(1/p_i + \log p_i)$, and indeed for large composites the overwhelming majority of progressions $《99.97\%》_{\mathrm{R}}《99.98\%》_{\mathrm{A}}$ will be handled in this manner.

3.3 Smallish Primes

For progressions with p_i close to s, $《256 < p_i < 5.2\cdot 10^5》_{\mathrm{R}}《256 < p_i < 4.2\cdot 10^6》_{\mathrm{A}}$, each processor can handle very few progressions because it can produce at most one emission triplet every 《2》 clock cycles. Thus, the amortized cost of the processor, memory control circuitry and buffers is very high. Moreover, such progression cause emissions so often that communicating their emissions to distant bus lines (which is necessary if the state of each progression is maintained

[11] The frequency of emissions is s/p_i, and each emission occupies some delivery cell for $\Theta(s)$ clock cycles. The last two terms are due to DRAM storage, and have very small constants.

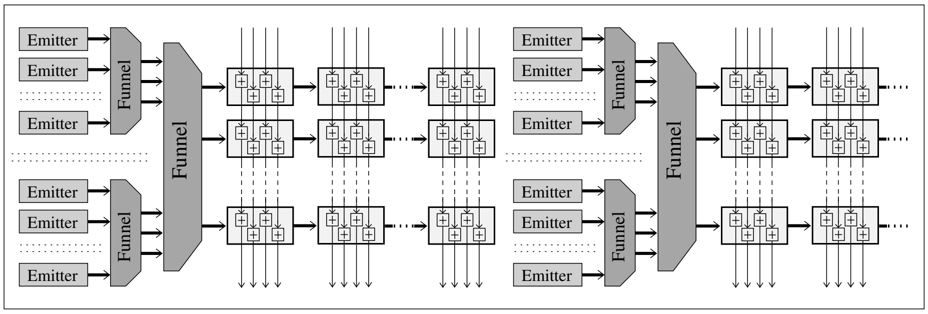

Fig. 3: Schematic structure of a smallish station.

at some single physical location) would involve enormous communication bandwidth. We thus introduce another station design, which differs in several ways from the largish stations (see Fig 3).

Emitters and funnels. The first change is to replace the combination of the processors, memory and buffers by other units. Delivery pairs are now created directly by *emitters*, which are small circuits that handle a single progression each (as in TWINKLE). An emitter maintains the state of the progression using internal registers, and occasionally emits delivery pairs of the form $(\lfloor \log p_i \rceil, \ell)$ which indicate that the value $\lfloor \log p_i \rceil$ should be added to the ℓ-th bus line some fixed time interval later. Appendix A.2 describes a compact emitters design.

Each emitter is continuously updating its internal counters, but it creates a delivery pair only once per roughly $\sqrt{p_i}$ (between $\langle\!\langle 8 \rangle\!\rangle_{\mathrm{R}}$ and $\langle\!\langle 512 \rangle\!\rangle_{\mathrm{R}}$ clock cycles — see below). It would be wasteful to connect each emitter to a dedicated delivery line. This is solved using *funnels*, which "compress" their sparse inputs as follows. A funnel has a large number of input lines, connected to the outputs of many adjacent emitters; we may think of it as receiving a sequence of one-dimensional arrays, most of whose elements are empty. The funnel outputs a sequence of much shorter arrays, whose non-empty elements are exactly the non-empty elements of the input array received a fixed number of clock cycle earlier. The funnel outputs are connected to the delivery lines. Appendix A.3 describes an implementation of funnels using modified shift registers.

Duplication. The other major change is duplication of the progression states, in order to move the sources of the delivery pairs closer to their destination and reduce the cross-bus communication bandwidth. Each progression is handled by $n_i \approx s/\sqrt{p_i}$ independent emitters[12] which are placed at regular intervals across the bus. Accordingly we fragment the delivery lines into segments that span $s/n_i \approx \sqrt{p_i}$ bus lines each. Each emitter is connected (via a funnel) to a different segment, and sends emissions to this segment every $p_i/sn_i \approx \sqrt{p}$ clock cycles. As emissions reach their destination quicker, we can decrease the total

[12] $\langle\!\langle n_i = s/2\sqrt{p_i} \rangle\!\rangle$ rounded to a power of 2 (cf. A.2), which is in the range $\langle\!\langle \{2, \ldots, 128\} \rangle\!\rangle_{\mathrm{R}}$.

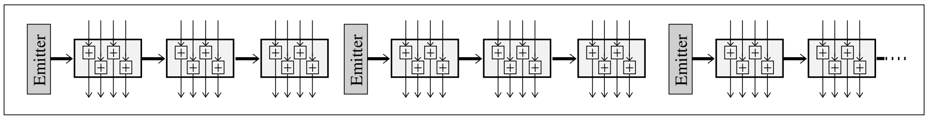

Fig. 4: Schematic structure of a tiny station, for a single progression.

number of delivery lines. Also, there is a corresponding decrease in the emission frequency of any specific emitter, which allows us to handle p_i close to (or even smaller than) s. Overall there are $\langle\!\langle 501 \rangle\!\rangle_{\mathrm{R}}$ delivery lines in the smallish stations, broken into segments of various sizes.

Notes. Asymptotically the amortized circuit area per smallish progression is $\Theta(\,(s/\sqrt{p_i}+1)\,(\log s+\log p_i)\,)$. The term 1 is less innocuous than it appears — it hides a large constant (roughly the size of an emitter plus the amortized funnel size), which dominates the cost for large p_i.

3.4 Tiny Primes

For very small primes, the amortized cost of the duplicated emitters, and in particular the related funnels, becomes too high. On the other hand, such progressions cause several emissions at every clock cycle, so it is less important to amortize the cost of delivery lines over several progressions. This leads to a third station design for the tiny primes $\langle\!\langle p_i < 256 \rangle\!\rangle$. While there are few such progressions, their contributions are significant due to their very small periods.

Each tiny progression is handled independently, using a dedicated delivery line. The delivery line is partitioned into segments of size somewhat smaller than p_i,[13] and an emitter is placed at the input of each segment, without an intermediate funnel (see Fig 4). These emitters are a degenerate form of the ones used for smallish progressions (cf. A.2). Here we cannot interleave the adders in delivery cells as done in largish and smallish stations, but the carry-save adders are smaller since they only (conditionally) add the small constant $\lfloor \log p_i \rceil$. Since the area occupied by each progression is dominated by the delivery lines, it is $\Theta(s)$ regardless of p_i.

Some additional design considerations are discussed in Appendix A.

4 Cost estimates

Having outlined the design and specified the problem size, we next estimate the cost of a hypothetical TWIRL device using today's VLSI technology. The hardware parameters used are specified in Appendix B.1. While we tried to produce realistic figures, we stress that these estimates are quite rough and rely on many approximations and assumptions. They should only be taken to indicate

[13] The segment length is the largest power of 2 smaller than p_i (cf. A.2).

the order of magnitude of the true cost. We have not done any detailed VLSI design, let alone actual implementation.

4.1 Cost of Sieving for 1024-bit Composites

We assume the following NFS parameters: $B_{\mathrm{R}} = 3.5 \cdot 10^9$, $B_{\mathrm{A}} = 2.6 \cdot 10^{10}$, $R = 1.1 \cdot 10^{15}$, $H \approx 2.7 \cdot 10^8$ (cf. B.2). We use the cascaded sieves variant of Appendix A.6.

For the rational side we set $s_R = 4{,}096$. One rational TWIRL device requires 15,960mm^2 of silicon wafer area, or 1/4 of a 30cm silicon wafer. Of this, 76% is occupied by the largish progressions (and specifically, 37% of the device is used for the DRAM banks), 21% is used by the smallish progressions and the rest (3%) is used by the tiny progressions. For the algebraic side we set $s_A = 32{,}768$. One algebraic TWIRL device requires 65,900mm^2 of silicon wafer area — a full wafer. Of this, 94% is occupied by the largish progressions (66% of the device is used for the DRAM banks) and 6% is used by the smallish progressions. Additional parameters of are mentioned throughout Section 3.

The devices are assembled in clusters that consist each of 8 rational TWIRLs and 1 algebraic TWIRL, where each rational TWIRL has a unidirectional link to the algebraic TWIRL over which it transmits 12 bits per clock cycle. A cluster occupies three wafers, and handles a full sieve line in R/s_A clock cycles, i.e., 33.4 seconds when clocked at 1GHz. The full sieving involves H sieve lines, which would require 194 years when using a single cluster (after the 33% saving of Appendix A.5.) At a cost of \$2.9M (assuming \$5,000 per wafer), we can build 194 independent TWIRL clusters that, when run in parallel, would complete the sieving task within 1 year.

After accounting for the cost of packaging, power supply and cooling systems, adding the cost of PCs for collecting the data and leaving a generous error margin,[14] it appears realistic that all the sieving required for factoring 1024-bit integers can be completed within 1 year by a device that cost \$10M to manufacture. In addition to this per-device cost, there would be an initial NRE cost on the order of \$20M (for design, simulation, mask creation, etc.).

4.2 Implications for 1024-bit Composites

It has been often claimed that 1024-bit RSA keys are safe for the next 15 to 20 years, since both NFS relation collection and the NFS matrix step would be unfeasible (e.g., [4, 21] and a NIST guideline draft [18]). Our evaluation suggests that sieving can be achieved within one year at a cost of \$10M (plus a one-time cost of \$20M), and recent works [16, 8] indicate that for our NFS parameters the matrix can also be performed at comparable costs.

[14] It is a common rule of thumb to estimate the total cost as twice the silicon cost; to be conservative, we triple it.

With efficient custom hardware for both sieving and the matrix step, other subtasks in the NFS algorithm may emerge as bottlenecks.[15] Also, our estimates are hypothetical and rely on numerous approximations; the only way to learn the precise costs involved would be to perform a factorization experiment.

Our results do not imply that breaking 1024-bit RSA is within reach of individual hackers. However, it is difficult to identify any specific issue that may prevent a sufficiently motivated and well-funded organization from applying the Number Field Sieve to 1024-bit composites within the next few years. This should be taken into account by anyone planning to use a 1024-bit RSA key.

4.3 Cost of Sieving for 512-bits Composites

Since several hardware designs [19, 15, 10, 7] were proposed for the sieving of 512-bit composites, it would be instructive to obtain cost estimates for TWIRL with the same problem parameters. We assume the same parameters as in [15, 7]: $B_{\mathrm{R}} = B_{\mathrm{A}} = 2^{24} \approx 1.7 \cdot 10^7$, $R = 1.8 \cdot 10^{10}, 2H = 1.8 \cdot 10^6$. We set $s = 1{,}024$ and use the same cost estimation expressions that lead to the 1024-bit estimates.

A single TWIRL device would have a die size of about 800mm^2, 56% of which are occupied by largish progressions and most of the rest occupied by smallish progressions. It would process a sieve line in 0.018 seconds, and can complete the sieving task within 6 hours.

For these NFS parameters TWINKLE would require 1.8 seconds per sieve line, the FPGA-based design of [10] would require about 10 seconds and the mesh-based design of [7] would require 0.36 seconds. To provide a fair comparison to TWINKLE and [7], we should consider a single wafer full of TWIRL devices running in parallel. Since we can fit 79 of them, the effective time per sieve line is 0.00022 seconds.

Thus, in factoring 512-bit composites the basic TWIRL design is about 1,600 times more cost effective than the best previously published design [7], and 8,100 times more cost effective than TWINKLE. Adjusting the NFS parameters to take advantage of the cascaded-sieves variant (cf. A.6) would further increase this gap. However, even when using the basic variant, a single wafer of TWIRLs can complete the sieving for 512-bit composites in under 10 minutes.

4.4 Cost of Sieving for 768-bits Composites

We assume the following NFS parameters: $B_{\mathrm{R}} = 1 \cdot 10^8$, $B_{\mathrm{A}} = 1 \cdot 10^9$, $R = 3.4 \cdot 10^{13}$, $H \approx 8.9 \cdot 10^6$ (cf. B.2). We use the cascaded sieves variant of Appendix A.6, with $s_R = 1{,}024$ and $s_A = 4{,}096$. For this choice, a rational sieve occupies 1,330mm^2 and an algebraic sieve occupies 4,430mm^2. A cluster consisting of 4 rational sieves and one algebraic sieve can process a sieve line in 8.3 seconds, and 6 independent clusters can fit on a single 30cm silicon wafer.

[15] Note that for our choice of parameters, the cofactor factorization is cheaper than the sieving (cf. Appendix A.7).

Thus, a single wafer of TWIRL clusters can complete the sieving task within 95 days. This wafer would cost about \$5,000 to manufacture — one tenth of the RSA-768 challenge prize [20].[16]

4.5 Larger composites

For largish progressions, the amortized cost per progression is $\Theta(s^2(\log s)/p_i + \log s + \log p_i)$ with small constants (cf. 3.2). For smallish progressions, the amortized cost is $\Theta(\,(s/\sqrt{p_i}+1)\,(\log s + \log p_i)\,)$ with much larger constants (cf. 3.3). For a serial implementation (PC-based or TWINKLE), the cost per progression is clearly $\Omega(\log p_i)$. This means that asymptotically we can choose $s = \tilde{\Theta}(\sqrt{B})$ to get a speed advantage of $\tilde{\Theta}(\sqrt{B})$ over serial implementations, while maintaining the small constants. Indeed, we can keep increasing s essentially for free until the area of the largish processors, buffers and delivery lines becomes comparable to the area occupied by the DRAM that holds the progression triplets.

For some range of input sizes, it may be beneficial to reduce the amount of DRAM used for largish progressions by storing only the prime p_i, and computing the rest of the progression triplet values on-the-fly in the special-purpose processors (this requires computing the roots modulo p_i of the relevant NFS polynomial).

If the device would exceed the capacity of a single silicon wafer, then as long as the bus itself is narrower than a wafer, we can (with appropriate partitioning) keep each station fully contained in some wafer; the wafers are connected in a serial chain, with the bus passing through all of them.

5 Conclusion

We have presented a new design for a custom-built sieving device. The device consists of a thick pipeline that carries sieve locations through thrilling adventures, where they experience the addition of progression contributions in myriad different ways that are optimized for various scales of progression periods. In factoring 512-bit integers, the new device is 1,600 times faster than best previously published designs. For 1024-bit composites and appropriate choice of NFS parameters, the new device can complete the sieving task within 1 year at a cost of \$10M, thereby raising some concerns about the security of 1024-bit RSA keys.

Acknowledgments. This work was inspired by Daniel J. Bernstein's insightful work on the NFS matrix step, and its adaptation to sieving by Willi Geiselmann and Rainer Steinwandt. We thank the latter for interesting discussions of their design and for suggesting an improvement to ours. We are indebted to Arjen K. Lenstra for many insightful discussions, and to Robert D. Silverman,

[16] Needless to say, this disregards an initial cost of about \$20M. This initial cost can be significantly reduced by using older technology, such as 0.25μm process, in exchange for some decrease in sieving throughput.

Andrew "bunnie" Huang and Michael Szydlo for valuable comments and suggestions. Early versions of [14] and the polynomial selection programs of Jens Franke and Thorsten Kleinjung were indispensable in obtaining refined estimates for the NFS parameters.

References

1. F. Bahr, J. Franke, T. Kleinjung, M. Lochter, M. Böhm, *RSA-160*, e-mail announcement, Apr. 2003, `http://www.loria.fr/~zimmerma/records/rsa160`
2. Daniel J. Bernstein, *How to find small factors of integers*, manuscript, 2000, `http://cr.yp.to/papers.html`
3. Daniel J. Bernstein, *Circuits for integer factorization: a proposal*, manuscript, 2001, `http://cr.yp.to/papers.html`
4. Richard P. Brent, *Recent progress and prospects for integer factorisation algorithms*, proc. COCOON 2000, LNCS **1858** 3–22, Springer-Verlag, 2000
5. S. Cavallar, B. Dodson, A.K. Lenstra, W. Lioen, P.L. Montgomery, B. Murphy, H.J.J. te Riele, et al., *Factorization of a 512-bit RSA modulus*, proc. Eurocrypt 2000, LNCS **1807** 1–17, Springer-Verlag, 2000
6. Don Coppersmith, *Modifications to the number field sieve*, Journal of Cryptology **6** 169–180, 1993
7. Willi Geiselmann, Rainer Steinwandt, *A dedicated sieving hardware*, proc. PKC 2003, LNCS **2567** 254–266, Springer-Verlag, 2002
8. Willi Geiselmann, Rainer Steinwandt, *Hardware to solve sparse systems of linear equations over GF(2)*, proc. CHES 2003, LNCS, Springer-Verlag, to be published.
9. International Technology Roadmap for Semiconductors 2001, `http://public.itrs.net/`
10. Hea Joung Kim, William H. Magione-Smith, *Factoring large numbers with programmable hardware*, proc. FPGA 2000, ACM, 2000
11. Robert Lambert, *Computational aspects of discrete logarithms*, Ph.D. Thesis, University of Waterloo, 1996
12. Arjen K. Lenstra, H.W. Lenstra, Jr., (eds.), *The development of the number field sieve*, Lecture Notes in Math. **1554**, Springer-Verlag, 1993
13. Arjen K. Lenstra, Bruce Dodson, *NFS with four large primes: an explosive experiment*, proc. Crypto '95, LNCS **963** 372–385, Springer-Verlag, 1995
14. Arjen K. Lenstra, Bruce Dodson, James Hughes, Paul Leyland, *Factoring estimates for 1024-bit RSA modulus*, to be published.
15. Arjen K. Lenstra, Adi Shamir, *Analysis and Optimization of the TWINKLE Factoring Device*, proc. Eurocrypt 2002, LNCS **1807** 35–52, Springer-Verlag, 2000
16. Arjen K. Lenstra, Adi Shamir, Jim Tomlinson, Eran Tromer, *Analysis of Bernstein's factorization circuit*, proc. Asiacrypt 2002, LNCS **2501** 1–26, Springer-Verlag, 2002
17. Brian Murphy, *Polynomial selection for the number field sieve integer factorization algorithm*, Ph. D. thesis, Australian National University, 1999
18. National Institute of Standards and Technology, *Key management guidelines, Part 1: General guidance (draft)*, Jan. 2003, `http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html`
19. Adi Shamir, *Factoring large numbers with the TWINKLE device (extended abstract)*, proc. CHES'99, LNCS **1717** 2–12, Springer-Verlag, 1999

20. RSA Security, *The new RSA factoring challenge*, web page, Jan. 2003, `http://www.rsasecurity.com/rsalabs/challenges/factoring/`
21. Robert D. Silverman, *A cost-based security analysis of symmetric and asymmetric key lengths*, Bulletin 13, RSA Security, 2000, `http://www.rsasecurity.com/rsalabs/bulletins/bulletin13.html`
22. Web page for this paper, `http://www.wisdom.weizmann.ac.il/~tromer/twirl`

A Additional Design Considerations

A.1 Delivery Lines

The delivery lines are used by all station types to carry delivery pairs from their source (buffer, funnel or emitter) to their destination bus line. Their basic structure is described in Section 3.2. We now describe methods for implementing them efficiently.

Interleaving. Most of the time the cells in a delivery line act as shift registers, and their adders are unused. Thus, we can reduce the cost of adders and registers by interleaving. We use larger delivery cells that span r $\langle\!\langle = 4 \rangle\!\rangle_{\mathrm{R}}$ adjacent bus lines, and contain an adder just for the q-th line among these, with q fixed throughout the delivery line and incremented cyclically in the subsequent delivery lines. As a bonus, we now put every r adjacent delivery lines in a single bus pipeline stage, so that it contains one adder per bus line. This reducing the number of bus pipelining registers by a factor of r throughout the largish stations.

Since the emission pairs traverse the delivery lines at a rate of r lines per clock cycle, we need to skew the space-time assignment of sieve locations so that as distance from the buffer to the bus line increases, the "age" $\lfloor a/s \rfloor$ of the sieve locations decreases. More explicitly: at time t, sieve location a is handled by the $\lfloor (a \bmod s)/r \rfloor$-th cell[17] of one of the r delivery lines at stage $t - \lfloor a/sr \rfloor - \lfloor (a \bmod s)/r \rfloor$ of the bus pipeline, if it exists.

In the largish stations, the buffer is entrusted with the role of sending delivery pairs to delivery lines that have an adder at the appropriate bus line; an improvement by a factor of 2 is achieved by placing the buffers at the middle of the bus, with the two halves of each delivery line directed outwards from the buffer. In the smallish and tiny stations we do not use interleaving.

Note that whenever we place pipelining registers on the bus, we must delay all downstream delivery lines connected to this buffer by a clock cycle. This can be done by adding pipeline stages at the beginning of these delivery lines.

Carry-save adders. Logically, each bus line carries a $\log_2 T$ $\langle\!\langle = 10 \rangle\!\rangle$-bit integer. These are encoded by a redundant representation, as a pair of $\log_2 T$-bit integers whose sum equals the sum of the $\lfloor \log p_i \rceil$ contributions so far. The additions at the delivery cells are done using carry-save adders, which have inputs a,b,c and whose output is a representation of the sum of their inputs in the form of a pair e,f such that $e + f = a + b + c$. Carry-save adders are very compact and support a high

[17] After the change made in Appendix A.2 this becomes $\lfloor \mathrm{rev}(a \bmod s)/r \rfloor$, where $\mathrm{rev}(\cdot)$ denotes bit-reversal of $\log_2 s$-bit numbers and s,r are powers of 2.

clock rate, since they do not propagate carries across more than one bit position. Their main disadvantage is that it is inconvenient to perform other operations directly on the redundant representation, but in our application we only need to perform a long sequence of additions followed by a single comparison at the end. The extra bus wires due to the redundant representation can be accommodated using multiple metal layers of the silicon wafer.[18]

To prevent wrap-around due to overflow when the sum of contributions is much larger than T, we slightly alter the carry-save adders by making their most significant bits "sticky": once the MSBs of both values in the redundant representation become 1 (in which case the sum is at least T), further additions do not switch them back to 0.

A.2 Implementation of Emitters

The designs of smallish and tiny progressions (cf. 3.3, 3.4) included *emitter* elements. An emitter handles a single progression P_i, and its role is to emit the delivery pairs $(\lfloor \log p_i \rceil, \ell)$ addressed to a certain group G of adjacent lines, $\ell \in G$. This subsection describes our proposed emitter implementation. For context, we first describe some less efficient designs.

Straightforward implementations. One simple implementation would be to keep a $\lceil \log_2 p_i \rceil$-bit register and increment it by s modulo p_i every clock cycle. Whenever a wrap-around occurs (i.e., this progression causes an emission), compute ℓ and check if $\ell \in G$. Since the register must be updated within one clock cycle, this requires an expensive carry-lookahead adder. Moreover, if s and $|G|$ are chosen arbitrarily then calculating ℓ and testing whether $\ell \in G$ may also be expensive. Choosing s, $|G|$ as power of 2 reduces the costs somewhat.

A different approach would be to keep a counter that counts down the time to the next emission, as in [19], and another register that keeps track of ℓ. This has two variants. If the countdown is to the next emission of this triplet regardless of its destination bus line, then these events would occur very often and again require low-latency circuitry (also, this cannot handle $p_i < s$). If the countdown is to the next emission into G, we encounter the following problem: for any set G of bus lines corresponding to adjacent residues modulo s, the intervals at which P_i has emissions into G are irregular, and would require expensive circuitry to compute.

Line address bit reversal. To solve the last problem described above and use the second countdown-based approach, we note the following: the assignment of sieve locations to bus lines (within a clock cycle) can be done arbitrarily, but the partition of wires into groups G should be done according to physical proximity. Thus, we use the following trick. Choose $s = 2^\alpha$ and $|G| = 2^{\beta_i} \approx \sqrt{p_i}$ for some integers $\alpha \langle\!\langle = 12 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle = 15 \rangle\!\rangle_{\mathrm{A}}$ and β_i. The residues modulo s are assigned to bus lines with bit-reversed indices; that is, sieve locations congruent to w modulo s

[18] Should this prove problematic, we can use the standard integer representation with carry-lookahead adders, at some cost in circuit area and clock rate.

are handled by the bus line at physical location $\mathrm{rev}(w)$, where

$$w = \sum_{i=0}^{\alpha-1} c_i 2^i \ , \quad \mathrm{rev}(w) = \sum_{i=0}^{\alpha-1} c_{\alpha-1-i} 2^i \quad \text{for some } c_0, \ldots, c_{\alpha-1} \in \{0,1\}$$

The j-th emitter of the progression P_i, $j \in \{0,\ldots,2^{\alpha-\beta_i}\}$, is in charge of the j-th group of 2^{β_i} bus lines. The advantage of this choice is the following.

Lemma 1. *For any fixed progression with $p_i > 2$, the emissions destined to any fixed group occur at regular time intervals of $T_i = \lfloor 2^{-\beta_i} p_i \rfloor$, up to an occasional delay of one clock cycle due to modulo s effects.*

Proof. Emissions into the j-th group correspond to sieve locations $a \in P_i$ that fulfill $\lfloor \mathrm{rev}(a \bmod s)/2^{\beta_i} \rfloor = j$, which is equivalent to $a \equiv c_j \,(\mathrm{mod}\ 2^{\alpha-\beta_i})$ for some c_j. Since $a \in P_i$ means $a \equiv r_i \,(\mathrm{mod}\ p_i)$ and p_i is coprime to $2^{\alpha-\beta_i}$, by the Chinese Remainder Theorem we get that the set of such sieve locations is exactly $P_{i,j} \equiv \{a : a \equiv c_{i,j} \,(\mathrm{mod}\ 2^{\alpha-\beta_i} p_i)\}$ for some $c_{i,j}$. Thus, a pair of consecutive $a_1,a_2 \in P_{i,j}$ fulfill $a_2 - a_1 = 2^{\alpha-\beta_i} p_i$. The time difference between the corresponding emissions is $\Delta = \lfloor a_2/s \rfloor - \lfloor a_1/s \rfloor$. If $(a_2 \bmod s) > (a_1 \bmod s)$ then $\Delta = \lfloor (a_2 - a_1)/s \rfloor = \lfloor 2^{\alpha-\beta_i} p_i / s \rfloor = T_i$. Otherwise, $\Delta = \lceil (a_2 - a_1)/s \rceil = T_i + 1$. $\square$

Note that $T_i \approx \sqrt{p_i}$, by the choice of β_i.

Emitter structure. In the smallish stations, each emitter consists of two counters, as follows.

- Counter A operates modulo $T_i = \lfloor 2^{-\beta_i} p_i \rfloor$ (typically $\langle\!\langle 7 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 5 \rangle\!\rangle_{\mathrm{A}}$ bits), and keeps track of the time until the next emission of this emitter. It is decremented by 1 (nearly) every clock cycle.
- Counter B operates modulo 2^{β_i} (typically $\langle\!\langle 10 \rangle\!\rangle_{\mathrm{R}} \langle\!\langle 15 \rangle\!\rangle_{\mathrm{A}}$ bits). It keeps track of the β_i most significant bits of the residue class modulo s of the sieve location corresponding to the next emission. It is incremented by $2^{\alpha-\beta_i} p_i \bmod 2^{\beta_i}$ whenever Counter A wraps around. Whenever Counter B wraps around, Counter A is suspended for one clock cycle (this corrects for the modulo s effect).

A delivery pair $(\lfloor \log p_i \rceil, \ell)$ is emitted when Counter A wraps around, where $\lfloor \log p_i \rceil$ is fixed for each emitter. The target bus line ℓ gets β_i of its bits from Counter B. The $\alpha - \beta_i$ least significant bits of ℓ are fixed for this emitter, and they are also fixed throughout the relevant segment of the delivery line so there is no need to transmit them explicitly.

The physical location of the emitter is near (or underneath) the group of bus lines to which it is attached. The counters and constants need to be set appropriately during device initialization. Note that if the device is custom-built for a specific factorization task then the circuit size can be reduced by hard-wiring many of these values[19]. The combined length of the counters is roughly

[19] For sieving the rational side of NFS, it suffices to fix the smoothness bounds. Similarly for the preprocessing stage of Coppersmith's Factorization Factory [6] .

$\log_2 p_i$ bits, and with appropriate adjustments they can be implemented using compact ripple adders[20] as in [15].

Emitters for tiny progressions. For tiny stations, we use a very similar design. The bus lines are again assigned to residues modulo s in bit-reversed order (indeed, it would be quite expensive to reorder them). This time we choose β_i such that $|G| = 2^{\beta_i}$ is the largest power of 2 that is smaller than p_i. This fixes $T_i = 1$, i.e., an emission occurs every one or two clock cycles. The emitter circuitry is identical to the above; note that Counter A has become zero-sized (i.e., a wire), which leaves a single counter of size $\beta_i \approx \log_2 p_i$ bits.

A.3 Implementation of Funnels

The smallish stations use *funnels* to compact the sparse outputs of emitters before they are passed to delivery lines (cf. 3.3). We implement these funnels as follows.

An n-to-m funnel ($n \gg m$) consists of a matrix of n columns and m rows, where each cell contains registers for storing a single progression triplet. At every clock cycle inputs are fed directly into the top row, one input per column, scheduled such that the i-th element of the t-th input array is inserted into the i-th column at time $t+i$. At each clock cycle, all values are shifted horizontally one column to the right. Also, each value is shifted one row down if this would not overwrite another value. The t-th output array is read off the rightmost column at time $t+n$.

For any $m < n$ there is some probability of "overflow" (i.e., insertion of input value into a full column). Assuming that each input is non-empty with probability ν independently of the others ($\nu \approx 1/\sqrt{p_i}$; cf. 3.3), the probability that a non-empty input will be lost due to overflow is:

$$\sum_{k=m+1}^{n} \binom{n}{k} \nu^k (1-\nu)^{n-k} (k-m)/k$$

We use funnels with $\langle\!\langle m = 5 \rangle\!\rangle_{\mathrm{R}}$ rows and $\langle\!\langle n \approx 1/\nu \rangle\!\rangle_{\mathrm{R}}$ columns. For this choice and within the range of smallish progressions, the above failure probability is less than 0.00011. This certainly suffices for our application.

The above funnels have a suboptimal compression ratio n/m $\langle\!\langle \approx 1/5\nu \rangle\!\rangle_{\mathrm{R}}$, i.e., the probability ν' $\langle\!\langle \approx 1/5 \rangle\!\rangle_{\mathrm{R}}$ of a funnel output value being non-empty is still rather low. We thus feed these output into a second-level funnel $\langle\!\langle$with $m' = 35$, $n' = 14\rangle\!\rangle_{\mathrm{R}}$, whose overflow probability is less than 0.00016, and whose cost is amortized over many progressions. The output of the second-level funnel is fed into the delivery lines. The combined compression ratio of the two funnel levels is suboptimal by a factor of $5 \cdot 14/34 = 2$, so the number of delivery lines is twice the naive optimum. We do not interleave the adders in the delivery lines as done for largish stations (cf. A.1), in order to avoid the overhead of directing delivery pairs to an appropriate delivery line.[21]

[20] This requires insertion of small delays and tweaking the constant values.

[21] Still, the number of adders can be reduced by attaching a single adder to several bus lines using multiplexers. This may impact the clock rate.

A.4 Initialization

The device initialization consists of loading the progression states and initial counter values into all stations, and loading instructions into the bus bypass re-routing switches (after mapping out the defects).

The progressions differ between sieving runs, but reloading the device would require significant time (in [19] this became a bottleneck). We can avoid this by noting, as in [7], that the instances of sieving problem that occur in the NFS are strongly related, and all that is needed is to increase each r_i value by some constant value $\tilde{r}_i$ after each run. The $\tilde{r}_i$ values can be stored compactly in DRAM using $\log_2 p_i$ bits per progression (this is included in our cost estimates) and the addition can be done efficiently using on-wafer special-purpose processors. Since the interval R/s between updates is very large, we don't need to dedicate significant resources to performing the update quickly. For lattice sieving the situation is somewhat different (cf. A.8).

A.5 Eliminating Sieve Locations

In the NFS relation collection, we are only interesting in sieve locations a on the b-th sieve line for which $\gcd(a',b) = 1$ where $a' = a - R/2$, as other locations yield duplicate relations. The latter are eliminated by the candidate testing, but the sieving work can be reduced by avoiding sieve locations with $c \,|\, a',b$ for very small c. All software-based sievers consider the case $2 \,|\, a',b$ — this eliminates 25% of the sieve locations. In TWIRL we do the same: first we sieve normally over all the odd lines, $b \equiv 1 \pmod 2$. Then we sieve over the even lines, and consider only odd a' values; since a progression with $p_i > 2$ hits every p_i-th odd sieve location, the only change required is in the initial values loaded into the memories and counters. Sieving of these odd lines takes half the time compared to even lines.

We also consider the case $3 \,|\, a',b$, similarly to the above. Combining the two, we get four types of sieve runs: full-, half-, third- and sixth-length runs, for $b \bmod 6$ in $\{1,5\}$, $\{2,4\}$, $\{3\}$ and $\{0\}$ respectively. Overall, we get a 33% time reduction, essentially for free. It is not worthwhile to consider $c \,|\, a',b$ for $c > 3$.

A.6 Cascading the Sieves

Recall that the instances of the sieving problem come in pairs of *rational* and *algebraic* sieves, and we are interested in the a values that passed both sieves (cf. 2.1). However, the situation is not symmetric: B_{R} $\langle\!\langle 2.6 \cdot 10^{10} \rangle\!\rangle_{\mathrm{A}}$ is much larger than B_{R} $\langle\!\langle = 3.5 \cdot 10^{9} \rangle\!\rangle_{\mathrm{R}}$.[22] Therefore the cost of the algebraic sieves would dominate the total cost when s is chosen optimally for each sieve type. Moreover, for 1024-bit composites and the parameters we consider (cf. Appendix B), we cannot make the algebraic-side s as large as we wish because this would exceed the capacity of a single silicon wafer. The following shows a way to address this.

[22] B_{A} and B_{R} are chosen as to produce a sufficient probability of semi-smoothness for the values over which we are (implicitly) sieving: circa $\langle\!\langle 10^{101} \rangle\!\rangle_{\mathrm{A}}$ vs. circa $\langle\!\langle 10^{64} \rangle\!\rangle_{\mathrm{R}}$.

Let s_R and s_A denote the s values of the rational and algebraic sieves respectively. The reason we cannot increase s_A and gain further "free" parallelism is that the bus becomes unmanageably wide and the delivery lines become numerous and long (their cost is $\tilde{\Theta}(s^2)$). However, the bus is designed to sieve s_A sieve locations per pipeline stage. If we first execute the rational sieve then most of these sieve locations can be ruled out in advance: all but a small fraction $\langle\!\langle 1.7 \cdot 10^{-4} \rangle\!\rangle$ of the sieve locations do not pass the threshold in the rational sieve,[23] and thus cannot form candidates regardless of their algebraic-side quality.

Accordingly, we make the following change in the design of algebraic sieves. Instead of a wide bus consisting of s_A lines that are permanently assigned to residues modulo s_A, we use a much narrower bus consisting of only $u \,\langle\!\langle = 32 \rangle\!\rangle_{\mathrm{A}}$ lines, where each line contains a pair (C,L). $L = (a \bmod s_A)$ identifies the sieve location, and C is the sum of $\lfloor \log p_i \rceil$ contributions to a so far. The sieve locations are still scanned in a pipelined manner at a rate of s_A locations per clock cycle, and all delivery pairs are generated as before at the respective units.

The delivery lines are different: instead of being long and "dumb", they are now short and "smart". When a delivery pair $(\lfloor \log p_i \rceil, \ell)$ is generated, ℓ is compared to L for each of the u lines (at the respective pipeline stage) in a single clock cycle. If a match is found, $\lfloor \log p_i \rceil$ is added to the C of that line. Otherwise (i.e., in the overwhelming majority of cases), the delivery pair is discarded.

At the head of the bus, we input pairs $(0, a \bmod s_A)$ for the sieve locations a that passed the rational sieve. To achieve this we wire the outputs of rational sieves to inputs of algebraic sieves, and operate them in a synchronized manner (with the necessary phase shift). Due to the mismatch in s values, we connect s_A/s_B rational sieves to each algebraic sieves. Each such cluster of s_A/s_B+1 sieving devices is jointly applied to one single sieve line at a time, in a synchronized manner. To divide the work between the multiple rational sieves, we use interleaving of sieve locations (similarly to the bit-reversal technique of A.2). Each rational-to-algebraic connection transmits at most one value of size $\log_2 s_R \,\langle\!\langle 12 \rangle\!\rangle$ bits per clock cycle (appropriate buffering is used to average away congestions).

This change greatly reduces the circuit area occupied by the bus wiring and delivery lines; for our choice of parameters, it becomes insignificant. Also, there is no longer need to duplicate emitters for smallish progressions (except when $p_i < s$). This allows us to use a large $s \,\langle\!\langle = 32{,}768 \rangle\!\rangle_{\mathrm{A}}$ for the algebraic sieves, thereby reducing their cost to less than that of the rational sieve (cf. 4.1). Moreover, it lets us further increase B_{A} with little effect on cost, which (due to tradeoffs in the NFS parameter choice) reduces H and R.

A.7 Testing Candidates

Having computed approximations of the sum of logarithms $g(a)$ for each sieve location a, we need to identify the resulting candidates, compute the corresponding sets $\{i : a \in P_i\}$, and perform some additional tests (cf. 2.1). These are implemented as follows.

[23] Before the cofactor factorization. Slightly more when $\lfloor \log p_i \rceil$ rounding is considered.

Identifying candidates. In each TWIRL device, at the end of the bus (i.e., downstream for all stations) we place an array of comparators, one per bus line, that identify a values for which $g(a) > T$. In the basic TWIRL design, we operate a pair of sieves (one rational and one algebraic) in unison: at each clock cycle, the sets of bus lines that passed the comparator threshold are communicated between the two devices, and their intersection (i.e., the candidates) are identified. In the cascaded sieves variant, only sieve locations that passed the threshold on the rational TWIRL are further processed by the algebraic TWIRL, and thus the candidates are exactly those sieve locations that passed the threshold in the algebraic TWIRL. The fraction of sieve locations that constitute candidates is very small $\langle\!\langle 2 \cdot 10^{-11} \rangle\!\rangle$.

Finding the corresponding progressions. For each candidate we need to compute the set $\{i : a \in P_i\}$, separately for the rational and algebraic sieves. From the context in the NFS algorithm it follows that the elements of this set for which p_i is relatively small can be found easily.[24] It thus appears sufficient to find the subset $\{i : a \in P_i\ ,\ p_i \text{ is largish}\}$, which is accomplished by having largish stations remember the p_i values of recent progressions and report them upon request.

To implement this, we add two dedicated pipelined channels passing through all the processors in the largish stations. The *lines channel*, of width $\log_2 s$ bits, goes upstream (i.e., opposite to the flow of values in the bus) from the threshold comparators. The *divisors channel*, of width $\log_2 B$ bits, goes downstream. Both have a pipeline register after each processor, and both end up as outputs of the TWIRL device. To each largish processor we attach a *diary*, which is a cyclic list of $\log_2 B$-bit values. Every clock cycle, the processor writes a value to its diary: if the processor inserted an emission triplet $(\lfloor \log p_i \rceil, \ell_i, \tau_i)$ into the buffer at this clock cycle, it writes the triple (p_i, ℓ_i, τ_i) to the diary; otherwise it writes a designated NULL value. When a candidate is identified at some bus line ℓ, the value ℓ is sent upstream through the lines channel. Whenever a processor sees an ℓ value on the lines channel, it inspects its diaries to see whether it made an emission that was added to bus line ℓ exactly z clock cycles ago, where z is the distance (in pipeline stages) from the processor's output into the buffer, through the bus and threshold comparators and back to the processor through the lines channel. This inspection is done by searching the $\langle\!\langle 64 \rangle\!\rangle$ diary entries preceeding the one written z clock cycles ago for a non-NULL value (p_i, ℓ_i) with $\ell_i = \ell$. If such a diary entry is found, the processor transmits p_i downstream via the divisors channel (with retry in case of collision). The probability of intermingling data belonging to different candidates is negligible, and even then we can recover (by appropriate divisibility tests).

In the cascaded sieves variant, the algebraic sieve records to diaries only those contributions that were not discarded at the delivery lines. The rational diaries are rather large ($\langle\!\langle 13{,}530 \rangle\!\rangle_{\mathrm{R}}$ entries) since they need to keep their entries a long time — the latency z includes passing through (at worst) all rational

[24] Namely, by finding the small factors of $F_j(a - R{,}b)$ where F_j is the relevant NFS polynomial and b is the line being sieved.

bus pipeline stages, all algebraic bus pipeline stages and then going upstream through all rational stations. However, these diaries can be implemented very efficiently as DRAM banks of a degenerate form with a fixed cyclic access order (similarly to the memory banks of the largish stations).

Testing candidates. Given the above information, the candidates have to be further processed to account for the various approximations and errors in sieving, and to account for the NFS "large primes" (cf. 2.1). The first steps (computing the values of the polynomials, dividing out small factors and the diary reports, and testing size and primality of remaining cofactors) can be effectively handled by special-purpose processors and pipelines, which are similar to the division pipeline of [7, Section 4] except that here we have far fewer candidates (cf. C).

Cofactor factorization. The candidates that survived the above steps (and whose cofactors were not prime or sufficiently small) undergo cofactor factorization. This involves factorization of one (and seldom two) integers of size at most $\langle\!\langle 1 \cdot 10^{24} \rangle\!\rangle$. Less than $\langle\!\langle 2 \cdot 10^{-11} \rangle\!\rangle$ of the sieve locations reach this stage (this takes $\lfloor \log p_i \rceil$ rounding errors into consideration), and a modern general-purpose processor can handle each in less than 0.05 seconds. Thus, using dedicated hardware this can be performed at a small fraction of the cost of sieving. Also, certain algorithmic improvements may be applicable [2].

A.8 Lattice Sieving

The above is motivated by NFS line sieving, which has very large sieve line length R. Lattice sieving (i.e., "special-q") involves fewer sieving locations. However, lattice sieving has very short sieving lines (8192 in [5]), so the natural mapping to the lattice problem as defined here (i.e., lattice sieving by lines) leads to values of R that are too small.

We can adapt TWIRL to efficient lattice sieving as follows. Choose s equal to the width of the lattice sieving region (they are of comparable magnitude); a full lattice line is handled at each clock cycle, and R is the total number of points in the sieved lattice block. The definition (p_i,r_i) is different in this case — they are now related to the vectors used in lattice sieving by vectors (before they are lattice-reduced). The handling of modulo s wrap-around of progressions is now somewhat more complicated, and the emission calculation logic in all station types needs to be adapted. Note that the largish processors are essentially performing lattice sieving by vectors, as they are "throwing" values far into the "future", not to be seen again until their next emission event.

Re-initialization is needed only when the special-q lattices are changed (every $8192 \cdot 5000$ sieve locations in [5]), but is more expensive. Given the benefits of lattice sieving, it may be advantageous to use faster (but larger) re-initialization circuits and to increase the sieving regions (despite the lower yield); this requires further exploration.

A.9 Fault Tolerance

Due to its size, each TWIRL device is likely to have multiple local defects caused by imperfections in the VLSI process. To increase the yield of good devices, we make the following adaptations.

If any component of a station is defective, we simply avoid using this station. Using a small number of spare stations of each type (with their constants stored in reloadable latches), we can handle the corresponding progressions.

Since our device uses an addition pipeline, it is highly sensitive to faults in the bus lines or associated adders. To handle these, we can add a small number of spare line segments along the bus, and logically re-route portions of bus lines through the spare segments in order to bypass local faults. In this case, the special-purpose processors in largish stations can easily change the bus destination addresses (i.e., ℓ value of emission triplets) to account for re-routing. For smallish and tiny stations it appears harder to account for re-routing, so we just give up adding the corresponding $\lfloor \log p_i \rceil$ values; we may partially compensate by adding a small constant value to the re-routed bus lines. Since the sieving step is intended only as a fairly crude (though highly effective) filter, a few false-positives or false-negatives are acceptable.

B Parameters for Cost Estimates

B.1 Hardware

The hardware parameters used are those given in [16] (which are consistent with [9]): standard 30cm silicon wafers with 0.13μm process technology, at an assumed cost of \$5,000 per wafer. For 1024-bit and 768-bit composites we will use DRAM-type wafers, which we assume to have a transistor density of 2.8 μm^2 per transistor (averaged over the logic area) and DRAM density of $0.2\mu\text{m}^2$ per bit (averaged over the area of DRAM banks). For 512-bit composites we will use logic-type wafers, with transistor density of $2.38\mu\text{m}^2$ per transistor and DRAM density of $0.7\mu\text{m}^2$ per bit. The clock rate is 1GHz clock rate, which appears realistic with judicious pipelining of the processors.

We have derived rough estimates for all major components of the design; this required additional analysis, assumptions and simulation of the algorithms. Here are some highlights, for 1024-bit composites with the choice of parameters specified throughout Section 3. A typical largish special-purpose processor is assumed to require the area of $\langle\!\langle 96{,}400 \rangle\!\rangle_{\mathrm{R}}$ logic-density transistors (including the amortized buffer area and the small amount of cache memory, about $\langle\!\langle 14\text{Kbit} \rangle\!\rangle_{\mathrm{R}}$, that is independent of p_i). A typical emitter is assumed to require $\langle\!\langle 2{,}037 \rangle\!\rangle_{\mathrm{R}}$ transistors in a smallish station (including the amortized costs of funnels), and $\langle\!\langle 522 \rangle\!\rangle_{\mathrm{R}}$ in a tiny station. Delivery cells are assumed to require $\langle\!\langle 530 \rangle\!\rangle_{\mathrm{R}}$ transistors with interleaving (i.e., in largish stations) and $\langle\!\langle 1220 \rangle\!\rangle_{\mathrm{R}}$ without interleaving (i.e., in smallish and tiny stations). We assume that the memory system of Section 3.2 requires $\langle\!\langle 2.5 \rangle\!\rangle$ times more area per useful bit than standard DRAM, due to the required slack and and area of the cache. We assume that bus wires don't require

Table 1: Sieving parameters.

Parameter	Meaning	1024-bit	768-bit	512-bit
R	Width of sieve line	$1.1 \cdot 10^{15}$	$3.4 \cdot 10^{13}$	$1.8 \cdot 10^{10}$
H	Number of sieve lines	$2.7 \cdot 10^{8}$	$8.9 \cdot 10^{6}$	$9.0 \cdot 10^{5}$
B_{R}	Rational smoothness bound	$3.5 \cdot 10^{9}$	$1 \cdot 10^{8}$	$1.7 \cdot 10^{7}$
B_{A}	Algebraic smoothness bound	$2.6 \cdot 10^{10}$	$1 \cdot 10^{9}$	$1.7 \cdot 10^{7}$

wafer area apart from their pipelining registers, due to the availability of multiple metal layers. We take the cross-bus density of bus wires to be ⟪0.5⟫ bits per μm, possibly achieved by using multiple metal layers.

Note that since the device contains many interconnected units of non-uniform size, designing an efficient layout (which we have not done) is a non-trivial task. However, the number of different unit types is very small compared to designs that are commonly handled by the VLSI industry, and there is considerable room for variations. The mostly systolic design also enables the creation devices larger than the reticle size, using multiple steps of a single (or very few) mask set.

Using a fault-tolerant design (cf. A.9), the yield can made very high and functional testing can be done at a low cost after assembly. Also, the acceptable probability of undetected errors is much higher than that of most VLSI designs.

B.2 Sieving Parameters

To predict the cost of sieving, we need to estimate the relevant NFS parameters (R, H, B_{R}, B_{A}). The values we used are summarized in Table 1. The parameters for 512-bit composites are the same as those postulated for TWINKLE [15] and appear conservative compared to actual experiments [5].

To obtain reasonably accurate predictions for larger composites, we followed the approach of [14]; namely, we generated concrete pairs of NFS polynomials for the RSA-1024 and RSA-768 challenge composites [20] and estimated their relations yield. The search for NFS polynomials was done using programs written by Jens Franke and Thorsten Kleinjung (with minor adaptations). For our 1024-bit estimates we picked the following pair of polynomials, which have a common integer root modulo the RSA-1024 composite:

$$\begin{aligned}
f(x) = \quad & 1719304894236345143401011418080x^5 \\
& - 6991973488866605861074074186043634471x^4 \\
& + 27086030483569532894050974257851346649521314x^3 \\
& + 46937584052668574502886791835536552277410242359042x^2 \\
& - 101070294842572111371781458850696845877706899545394501384x \\
& - 22666915939490940578617524677045371189128909899716560398434136 \\
g(x) = \quad & 93877230837026306984571367477027x \\
& - 37934895496425027513691045755639637174211483324451628365
\end{aligned}$$

Subsequent analysis of relations yield was done by integrating the relevant smoothness probability functions [11] over the sieving region. Successful factorization requires finding sufficiently many *cycles* among the relations, and for two large primes per side (as we assumed) it is currently unknown how to predict the number of cycles from the number of relations, but we verified that the numbers appear “reasonable” compared to current experience with smaller composites. The 768-bit parameters were derived similarly. More details are available in a dedicated web page [22] and in [14].

Note that finding better polynomials will reduce the cost of sieving. Indeed, our algebraic-side polynomial is of degree 5 (due to a limitation of the programs we used), while there are theoretical and empirical reasons to believe that polynomials of somewhat higher degree can have significantly higher yield.

C Relation to Previous Works

TWINKLE. As is evident from the presentation, the new device shares with TWINKLE the property of time-space reversal compared to traditional sieving. TWIRL is obviously faster than TWINKLE, as two have comparable clock rates but the latter checks one sieve location per clock cycle whereas the former checks thousands. None the less, TWIRL is smaller than TWINKLE — this is due to the efficient parallelization and the use of compact DRAM storage for the largish progressions (it so happens that DRAM cannot be efficiently implemented on GaAs wafers, which are used by TWINKLE). We may consider using TWINKLE-like optical analog adders instead of electronic adder pipelines, but constructing a separate optical adder for each residue class modulo s would entail practical difficulties, and does not appear worthwhile as there are far fewer values to sum.

FPGA-based serial sieving. Kim and Mangione-Smith [10] describe a sieving device using off-the-shelf parts that may be only 6 times slower than TWINKLE. It uses classical sieving, without time-memory reversal. The speedup follows from increased memory bandwidth – there are several FPGA chips and each is connected to multiple SRAM chips. As presented this implementation does not rival the speed or cost of TWIRL. Moreover, since it is tied to a specific hardware platform, it is unclear how it scales to larger parallelism and larger sieving problems.

Low-memory sieving circuits. Bernstein [3] proposes to completely replace sieving by memory-efficient smoothness testing methods, such as the Elliptic Curve Method of factorization. This reduces the asymptotic time $\times$ space cost of the matrix step from $y^{3+o(1)}$ to $y^{2+o(1)}$, where y is subexponential in the length of the integer being factored and depends on the choice of NFS parameters. By comparison, TWIRL has a throughput cost of $y^{2.5+o(1)}$, because the speedup factor grows as the square root of the number of progressions (cf. 4.5). However, these asymptotic figures hide significant factors; based on current experience, for 1024-bit composites it appears unlikely that memory-efficient smoothness testing would rival the practical performance of traditional sieving, let alone that of TWIRL, in spite of its superior asymptotic complexity.

Mesh-based sieving. While [3] deals primarily with the NFS matrix step, it does mention "sieving via Schimmler's algorithm" and notes that its cost would be $L^{2.5+o(1)}$ (like TWIRL's). Geiselmann and Steinwandt [7] follow this approach and give a detailed design for a mesh-based sieving circuit. Compared to previous sieving devices, both [7] and TWIRL achieve a speedup factor of $\tilde{\Theta}(\sqrt{B})$.[25] However, there are significant differences in scalability and cost: TWIRL is 1,600 times more efficient for 512-bit composites, and ever more so for bigger composites or when using the cascaded sieves variant (cf. 4.3, A.6).

One reason is as follows. The mesh-based sorting of [7] is effective in terms of *latency*, which is why it was appropriate for the Bernstein's matrix-step device [3] where the input to each invocation depended on the output of the previous one. However, for sieving we care only about *throughput*. Disregarding latency leads to smaller circuits and higher clock rates. For example, TWIRL's delivery lines perform trivial one-dimensional unidirectional routing of values of size $\langle\!\langle 12+10 \rangle\!\rangle_{\mathrm{R}}$ bits, as opposed to complicated two-dimensional mesh sorting of progression states of size $\langle\!\langle 2 \cdot 31.7 \rangle\!\rangle_{\mathrm{R}}$.[26] For the algebraic sieves the situation is even more extreme (cf. A.6).

In the design of [7], the state of each progression is duplicated $\lceil \tilde{\Theta}(B/p_i) \rceil$ times (compared to $\lceil \tilde{\Theta}(\sqrt{B/p_i}) \rceil$ in TWIRL) or handled by other means; this greatly increases the cost. For the primary set of design parameters suggested in [7] for factoring 512-bit numbers, 75% of the mesh is occupied by duplicated values even though all primes smaller than 2^{17} are handled by other means: a separate division pipeline that tests potential candidates identified by the mesh, using over 12,000 expensive integer division units. Moreover, this assumes that the sums of $\lfloor \log p_i \rceil$ contributions from the progressions with $p_i > 2^{17}$ are sufficiently correlated with smoothness under all progressions; it is unclear whether this assumption scales.

TWIRL's handling of largish primes using DRAM storage greatly reduces the size of the circuit when implemented using current VLSI technology (90 DRAM bits vs. about 2500 transistors in [7]).

If the device must span multiple wafers, the inter-wafer bandwidth requirements of our design are much lower than that of [7] (as long as the bus is narrower than a wafer), and there is no algorithmic difficulty in handling the long latency of cross-wafer lines. Moreover, connecting wafers in a chain may be easier than connecting them in a 2D mesh, especially in regard to cooling and faults.

[25] Possibly less for [7] — an asymptotic analysis is lacking, especially in regard to the handling of small primes.

[26] The authors of [7] have suggested (in private communication) a variant of their device that routes emissions instead of sorting states, analogously to [16]. Still, mesh routing is more expensive than pipelined delivery lines.