Nice Attacks — but What is the Cost? Computational Models for Cryptanalysis

Charles Bouillaguet[1]

LIP6 laboratory, Sorbonne Université, Paris
France charles.bouillaguet@lip6.fr

Abstract. This paper discusses the implications of choosing a computational model to study the cost of cryptographic attacks and therefore quantify how dangerous they are. This choice is often unconscious and the chosen model itself is usually implicit; but it has repercussions on security evaluations.
We compare three reasonable computational models: *i*) the usual Random Access Machine (RAM) model; *ii*) the "Expensive Memory Model" explicitly introduced by several 3rd-round submissions to the Post-Quantum NIST competition (it states that a single access to a large memory costs as much as many local operations); *iii*) the venerable VLSI model using the Area-Time cost measure.
It is well-known that costs in the RAM model are lower that costs in the last two models. These have been claimed to be more realistic, and therefore to lead to more precise security evaluations. The main technical contribution of this paper is to show that the last two these models are incomparable. We identify a situation where the expensive memory model overestimates costs compared to the (presumably even more realistic) VLSI model.
In addition, optimizing the cost in each model is a distinct objective that leads to different attack parameters, and raises the question of what is the "best" way to proceed for an eventual attacker. We illustrate these discrepancies by studying several generic attacks against hash function and Feistel networks in the three models.

1 Introduction

Cryptographic schemes are used to enforce security properties (secrecy, integrity, etc.). Algorithms whose complete and successful execution would break security properties are cryptographic *attacks*. In most cases, security holds in a computational sense: attacks that would break security properties either require too many operations, or have a negligible probability of success.

"Too many operations" is often taken to a number that is believed to be out of reach of the day's computing capabilities. Presently, it is widely assumed that a computation requiring 2^{128} (classical) operations is intractable. More generally, it is expected that there is a fast-growing gap between the cost of cryptographic attacks and the cost of legitimate operations, as the size of the parameters increases. For instance, in symmetric cryptology, encryption schemes with n-bit

keys (resp. message authentication codes with n-bit tags) are often expected to provide "n bits of security", namely to resist to all attacks that require less than 2^n operations (legitimate operations only require polynomial time). It is often expected that brute-force is the best attack on secure symmetric schemes; any algorithm that improves over brute-force is a non-trivial cryptanalytic result.

Public-key cryptography necessarily relies on computational hardness assumptions: integer factoring, computing logarithms in some groups, finding short vectors in euclidean lattices, solving polynomial systems of equations, decoding random linear codes, etc. are assumed to be intractable for sufficiently large parameters sizes. In this context, exhaustive search is no longer the gold standard against which public-key primitives are measured.

Designers of concrete public-key primitives must choose key sizes. They have to make sure that the best known algorithms for the underlying hard problem would require too many operations (barring the invention of new, faster algorithms). This is sometimes a delicate issue, because it can be non-obvious to translate a rough complexity estimate into an actual number of operations (there can be hidden constants and even polynomial factors). This process has recently been at work in the context of the NIST Post-Quantum competition.

Studying the complexity of algorithms therefore plays an important role in cryptology. Cryptanalysts strive to invent attacks with lower and lower complexity, namely more and more dangerous and powerful attacks.

The number of elementary operations is the prime measure of complexity, in addition to estimating the amount of space and of "data" extracted from legitimate users. Inventing an attack that requires less operations that all previously known ones is usually considered an improvement over the state of the art. Improving the space or data complexity while not increasing the number of operations is also usually considered a net gain.

An attack against a symmetric cryptographic scheme requiring $2^{\frac{4}{5}n}$ operations and $2^{\frac{4}{5}n}$ bits of memory is usually considered an improvement over brute-force (2^n operations and no memory, with the same amount of data), because it uses less operations. Many such attacks have been published.

In any case, rigorously obtaining formulas for the complexity of an algorithm, either in a concrete or asymptotic sense, requires a *computational model*. The time complexity is usually defined to be the number of elementary operations in the model. Depending on the context, only certain operations may be accounted for: bit operations, group operations, block cipher evaluations, comparisons (for sorting), memory accesses, etc.

Besides this paper's technical contributions (outlined a bit later), the authors would like to offer several *positions* and submit them to a discussion within the cryptology community.

1.1 What is a Good Computational Model?

The Turing machine and the λ-calculus are two historical computational model that were used to formalize the concept of algorithm in the 1930's. They are still used in theoretical computer science, but are not very practical to write

programs for (it is known to be possible, but rarely ever done). In addition, they are quite unrelated to the actual hardware available at the present time. The ideal computational model should be simultaneously:

- As simple as possible (just simple operations).
- Sufficiently expressive (λ-reduction is not enough).
- Sufficiently abstract to shield us away from the intricacies of actual hardware ("*Starting from Ivy Bridge processors, there is an undocumented hardware next-page TLB prefetcher for virtual 4K pages*[1]").
- Sufficiently faithful to reality.

The first two points mean that translating high-level algorithmic ideas into a formal program in the computational model should be as easy as possible (ideally, it would be like using a well-known programming language). The third point means that studying the behavior of a program in the computational model should be as simple as possible. The fourth point means that the model is relevant because it correspond to possible computational systems in the material world.

These requirements imply several desirable features. For instance, an arithmetic operations between small integer should be an "elementary" operation, with the assumption that common hardware performs it in constant time. On the other hand, multi-precision arithmetic between million-bit numbers should not come for free.

The computational model most widely taught in all algorithmic classes at this moment is most likely the Random Access Machine (RAM) model or a variant thereof. It assumes that arithmetic operations and memory accesses are both elementary operations, but its exact features are often somewhat implicit and under-specified.

Position 1 *The Random Access Machine modem is well-suited to the study of* small *computations that take place in common, small computers and require only a modest amount of memory. Typically, when the data fit inside the memory of a single compute node, then it is true that the sequential running-time is well-correlated with the number of operations executed.*

1.2 Realism of Computational Models

Finding the "right" computational model is a long-standing, and still open question. Realism is probably the most contentious aspect.

A cryptographic primitive or protocol is considered secure if the best known attack is intractable. This means that running the attack should require more resources (time, memory, energy, ...) than available to any attacker. Security is therefore defined with respect to a computational model and a *cost function* that makes sense in this model. An attack is intractable is it costs too much, and its "threat level" is directly related to its cost in the computational model.

[1] This is true.

For instance, it is common to assume that the perspective of doing more than, say, 2^{128} bit operations, should deter any reasonable attacker. Therefore, taking the number of bit operations as a cost measure is relevant from a security point of view.

More generally speaking, to be useful from a security perspective, a computational model should have a cost function that relates to the material world. A high cost in the model must imply that actually running the computation on actual hardware is infeasible. It is strictly necessary that the model does not *overestimate* the difficulty of doing the actual computation, since this would render all theoretical security analyzes meaningless. The situation that must be avoided no matter what is that where running an attack is deemed intractable in the computational model but turns out to be feasible in the real world. Such a computational model would be unsuitable for theoretical reasoning in cryptology. Some early computational models (such as counter machines [34]) may overestimate actual costs.

In the context of cryptology, the faithfulness requirement stated above dictates that running a given computation in the model should not be harder than in the real world. But what if, on the contrary, the computational model *underestimates* the real world? It would make attacks appear more threatening in the model than what they could ever be in reality. From the point of view of security, this is not a problem; it is fine to be overly conservative. A dangerous attack (with low cost in the model) is a good enough reason to consider a cryptographic primitive and/or a set of parameters as "broken". It is better to be safe by discarding the scheme/parameters in face of the attack in the abstract computational model than to be sorry later when an actual break happens in the real world. The whole point of cryptanalytic research is to discover potential threats, early on if possible.

The obvious downside of a computational model that underestimates costs is that we may be worrying about unrealistic attacks that are *only* dangerous on fictitious abstract machines. These artificial attacks would nevertheless lead some cryptographic primitives to be dropped, or security parameters to be increased in some others, with an efficiency loss.

This is obviously undesirable and it can be related to the quest for tightness in security proofs. Tight security proofs are better than loose ones, because they allow for smaller key sizes and better efficiency with the same guaranteed security level. For the same reason, "tight", namely *more realistic* computational models are better.

Somewhat obviously, some attacks that are "valid" (cost less than the expected security level) in less realistic models are "invalid" (cost more than the expected security level) in more realistic computational models. These attacks are not necessarily bad. But attacks that are also valid in more realistic models are better, and potentially more dangerous.

Position 2 *Simple-but-unrealistic models of computations do not need to be abandoned. However, exploring the cost of cryptographic attacks in more realistic models results in a finer understanding of the danger they potentially cause.*

1.3 The Problem of Memory Accesses

One recurring realism issue concerns the cost of memory accesses. In the context of the NIST Post-Quantum competition, the designers of several third-round submissions observe that accessing an exponentially large memory is costly; it takes more time and it requires more energy than accessing a small memory. Attacks requiring an exponential number of accesses to an exponentially large memory should "cost more" than what the number of elementary operations in the RAM model suggests.

This observation allows the designers of these submissions to use slightly more aggressive parameter choices. Indeed, their claim is that some attacks would require a bit less operations than the security threshold, but that these attack in fact *cost more* because their perform a lot of expensive memory accesses.

For instance, the designers of Classic McEliece discuss the case of a parameter set named `mceliece6960119`, claimed to offer as much cryptographic resistance as a secure block cipher with 256-bit key. The designers discuss existing attacks, notably generic decoding algorithms for random linear codes, and state that:

> [...] *Subsequent ISD variants have reduced the number of bit operations considerably below* 2^{256}*. However none of these analyses took into account the costs of memory access.*
>
> (Classic McEliece 3rd-round submission [7], §8.2)

The designers of NTRU Prime distinguish attacks in "local" computational models, where information travels at finite speed (single-tape Turing machine, VLSI circuits, ...) and "non-local" ones where accessing memory is free. Comparing the energy consumption of a double-precision floating-point multiplication and that of a memory read on usual CPUs, the designers of `NTRU prime`

> *[...] estimate the cost of each access to a bit within* N *bits of memory as the cost of* $N^{0.5}/2^5$ *bit operations.*
>
> (NTRU Prime 3rd-round submission [8], §6.6)

These authors state that the usual computational model where accesses to an exponentially large memory are "free" is unrealistic (underestimates costs). They conclude that there are unrealistic attacks against their proposed scheme whose cost in the usual model is too low. On the other hand, in a presumably more realistic model where accessing a large memory is costly, all these attacks would cost more than the security threshold, and their scheme would be secure.

These arguments have then also been used by the Rainbow team [36]. It thus seems that a significant fraction of designers of public-key primitives adhere to this point of view.

This raises a wealth of interesting question. Is it true that accessing a large memory has cost equivalent to exponentially many local operations? Is the most common computational model too unrealistic? What are more realistic models? Are there attacks that are efficient only in unrealistic models? Should these attacks be considered as relevant? What are the implication on key sizes?

Falling short of an established name, we call the computational model described above the *Expensive Memory Model*. It asserts that accesses to a memory of size M have a cost equivalent to $\sqrt{M}$ "local" elementary operations that do not access memory. The complexity of an algorithm is then decomposed into the number of local operations and the number of memory accesses. The final cost is then expressed in terms of local operations, using the equivalence given above.

Position 3 *The expensive memory model is more realistic than the Random Access Machine model.*

1.4 The Problem of Counting the Number of Operations

There is another well-known problem with using the number of elementary operations as a cost function. This specific issue has been raised by Bernstein [5] and Aumasson [1], among others. Let us consider two algorithms both capable of breaking a symmetric primitive with n-bit keys:

Algorithm A Sophisticated. Requires $2^{\frac{3}{5}n}$ operations and $2^{\frac{3}{5}n}$ memory "cells".
Algorithm B Exhaustive search. Requires 2^n operations and little memory.

Just counting elementary operations, the sophisticated algorithm seems better; running it costs less than exhaustive search; it is a "more dangerous" attack. Incidentally, it is also better in the expensive memory model, because even if all operations are random memory accesses, they still cost only $2^{0.9n}$.

Running the sophisticated algorithm requires a machine of size $2^{\frac{3}{5}n}$ to hold its memory. Suppose that an adversary has enough resources to build a machine of this size. Then she also has the resources to build another machine that does exhaustive search in parallel using $2^{\frac{3}{5}n}$ processors (this trades storage hardware for computing hardware). Such a machine would terminate after $2^{\frac{2}{5}n}$ wall-clock time steps. This is less wall-clock time than a sequential implementation of the sophisticated algorithm, using a comparable amount of resources.

Counting the number of operations in the RAM model suggests that the sophisticated algorithm is better. However, a rational attacker would not even hesitate a second between implementing a *sequential* version of the sophisticated algorithm and a *parallel* implementation of exhaustive search. The latter is obviously a better use of resources, leading to a smaller time-to-solution using a comparable budget.

The comparison is, of course, unfair. Why would a rational attacker be restricted to sequential implementation of the sophisticated algorithm? Assume that she is capable of building a large machine with $2^{\frac{3}{5}n}$ memory cells *and* $2^{\frac{3}{5}n}$ processors (this machine has the same asymptotic size as before). Can the sophisticated algorithm be parallelized to run in less wall-clock time than exhaustive search on this large parallel machine? This would a strong incentive for the rational attacker to favor it over parallel exhaustive search. Counting the number of operations in a sequential computing model does not predict the outcome of this comparison, which nevertheless seems practically relevant.

In other terms, if an algorithm that competes against exhaustive search requires M memory cells, can it run on a parallel machine of size M in less than $2^n/M$ time steps? Exhaustive search does. This question is rarely, if ever, addressed by the authors of cryptographic attacks against symmetric schemes.

In fact, there is a more fundamental problem. Using the number of operations as the sole cost measure pushes algorithm designers to use as much memory as possible in order to reduce the number of operations. This leads to the design of (sequential) algorithms using a large memory that are even more likely to be less efficient than (parallel) exhaustive search because they require larger and larger machines to run. It is often unknown whether they can be parallelized efficiently. In addition, using a large memory means that there are most likely "hidden costs", as suggest by the discussion on the expensive memory model.

A model where a single sequential processor is attached to a large memory is, in fact, unrealistic. Such machines do not actually exist. Large memories are distributed over a large number of compute nodes connected by a network. For instance, the most powerful computer in the world at the time of this writing, the `fugaku` computer, has 150,000+ nodes with 32GB of memory, which makes about 2^{55} bits.

In addition, available memory is usually more restricted than available time. Take the case of the double-DES: a well-known meet-in-the-middle attack due to Diffie and Hellman breaks it in 2^{57} block-cipher evaluations using $\approx 2^{62}$ bits of memory [16]. The time complexity is not a problem (exhaustive search over the 56-bit keys has been done in the late 1990's) but the memory requirement makes this simple attack completely impractical.

Position 4 (controversial) *Counting the number of operations in the Random Access Machine Models pushes theoretically-minded cryptanalysts towards the invention of impractical attacks.*

1.5 Looking Back at a Venerable Computing Model

There are well-known computational models that do not have exhibit the problems discussed above. About 40 years ago, the VLSI computational model was widely studied. This models considers computer chips as graphs under some restrictions (components are laid on a two-dimensional grid; wires take space; wires are either horizontal or vertical; only two wires cross at a given point in space). Computation are necessary local operations (data must be moved to a common point in space to be acted upon).

A VLSI chip is characterized by its area A and the time T it takes to perform its task. If a specific computation has to be repeated many times (a common pattern in cryptographic attacks), then the AT product is the prime measure of efficiency: given some budget, there is an upper limit on the total circuit area that can be used. One can use a few large-but-fast circuits or many small-but-slow circuits. In the end the throughput is inversely proportional to AT. This strongly suggest to use the product AT as a cost measure. In addition, a machine of total size A, running for T time steps can perform at most AT operations.

In the sequel, we use the term "AT model" to denote the AT cost in the VLSI model.

The AT model satisfies the requirement that accessing a large memory should be costly: long wires are required to move the data around, and thus has repercussions on both area and time.

Some cryptographic constructions, such as memory-hard hash functions like `scrypt` explicitly state their security goals in the AT model. Several authors, including most notably Bernstein [4, 9, 6] and Wiener [42] advocated the use of the AT cost in cryptography.

Starting from 2012, Kleinjung, Lenstra, Page et Smart suggested to quantify the security level offered by several well-known cryptography primitives by estimating the amount of money an attacker would need to pay to a well-known public cloud operator to carry the attack [27]. These costs have been reevaluated every three years [15]. The cost model of a cloud operator is very close to the AT model: one pays proportionally to the amount of resource rented (A) and to the duration of the rental (T).

It must be noted that many large-scale cryptographic attacks have been carried out on a shared computing infrastructure (either public clouds [35] or public computation centers [11]). In this case, the cryptanalyst often has a "budget" given in CPU-hours — in other terms, an upper limit on the AT cost of the attack.

Many common operations have an AT cost asymptotically larger than the number of elementary operations they require ($n^{1.5}$ vs $n \log n$ for sorting). It is natural to expect that the cost of many cryptographic attacks is higher in the AT model than what their number of operations suggests. The AT cost of most complex computations is under-estimated by counting the number of elementary operations. At the very least, the AT costs seems to correlate with energy consumption, and someone has to pay the bill. Therefore, we are facing (at least) one of the two following situations:

1. Either the RAM model underestimates the cost of computation on actual hardware,
2. Or the AT model overestimates the cost of computation on actual hardware.

The first alternative seems much more likely to us.

Position 5 *The AT model is more realistic than the expensive memory model.*

1.6 Technical Contributions

This paper does not present new attacks. It discusses the cost of existing generic attacks (on symmetric constructions) in several computational models. In particular, we shows that:

- Judging an existing attack in the expensive memory model may be more complex than doing "[local operations] + [memory accesses] $\times \sqrt{\text{[Memory size]}}$".

Attacks can often be modified to reduce their cost in the expensive memory model. This may involve time-memory trade-offs (*more* time for *less* memory) or simply reorganizing the computation to reduce the number of memory accesses without altering the total number of operation. We show examples of the two cases using simple generic attacks on hash functions in sections 3 and 4.

- In some cases, the AT cost is necessarily (strictly) higher than the cost in the expensive memory model. We show an example with a simple generic second-preimage attack on hash functions in section 3.
- In some cases, the cost in the expensive memory model is apparently higher than the cost in the AT model, which seems counter-intuitive at first. We show an example with a simple key-collision search on HMAC in section 4. This is a red flag: the expensive memory model seems to overestimate actual costs. We exhibit a situation where the premise underlying the expensive memory model (access to a memory of size N costs as much as $\sqrt{N}$ local operations) is invalidated. As such, security analyses in the expensive memory model should be taken with a grain of salt.
- Optimizing the cost in the RAM model, the expensive memory model and the AT models are three different and incompatible objectives. This means that improving an attack in one model can make it worse in the other two. This is shown on an example in section 3.
- In particular, reducing the memory complexity without changing the number of operations, which is apparently a strict improvement, may lead to an increase of the cost in the expensive memory model, counter-intuitively. This happens when the improvement prevents time-memory trade-offs. An example is shown in section 5 with generic key-recovery attacks against Feistel networks.
- Somewhat obviously, complex generic attacks, such as the recent universal forgery attack against HMAC of Guo, Peyrin, Sasaki and Wang [24] can be better than brute-force in some simplified models and worse than brute-force in more realistic models (or even stop making sense at all). This attack is discussed in-depth in section 6.

All complexities stated in this paper are asymptotic. We always omit both constant factors and the "big O" notation.

2 Abstract Models of Computation

This section surveys the computational models discussed in the introduction.

2.1 The Random Access Machine

The Random Access Machine is one of the simplest and most likely the most widely-taught computational model. To the best of our knowledge, it has been introduced by Cook and Reckhow [13] in 1972–73 in order to model a Von

Neumann architecture. Informally speaking, it consists of a sequential processor connected to an arbitrarily large memory. A more thorough description is given by Van Emde Boas in the *Handbook of Theoretical Computer Science* [39].

The machine has a constant number of states, a (fixed-size) instruction pointer, a constant number of integer registers of *unbounded size*, an *infinite* number of memory cells $M[0], M[1], \ldots$, each capable of holding an integer of *unbounded size*. Here are the instructions that the machine can execute :

1. Control flow: **halt**, **goto**, **if** condition **then goto**. A condition is a test of the form $\texttt{register} == 0$ or $\texttt{register} \geq 0$.
2. Input/Output: **read** `register` and **print** `register`.
3. Data movement: $\texttt{register} \leftarrow i$, $\texttt{register} \leftarrow M[\texttt{register}]$, $M[\texttt{register}] \leftarrow \texttt{register}$, where i is an integer constant.
4. Boolean and arithmetic operations between registers.

The unbounded memory size is necessary to deal with arbitrarily large problem instances. In turn, this requires arbitrarily large registers to perform indirect memory accesses to the whole memory. Finally, unbounded memory cells are necessary to store pointers.

If addition and multiplication of registers are allowed as an "elementary operation", then the machine allows the execution of Shamir's factoring algorithm [33], which only uses a linear number of arithmetic operations. This is clearly not reasonable, therefore some restriction have to be enforced: preventing multiplication, restricting addition to incrementation, "billing" arithmetic operations more than $\mathcal{O}(1)$, etc. None of these solutions match the mental model that most programmers have of the complexity of their programs.

For instance, a function computed in $T(n)$ steps by a Random Access Machine able to add, substract, divide by two and compare to zero, can also be computed by a 7-tape Turing machine in $T(n)^3$ steps [29]. Disallowing multiplication between registers thus rules out the inadvertent polynomial-time integer factoring algorithm. However, this model is still problematic for many reasons; most programmer expect constant-time arithmetic operations and fixed-size memory cells.

2.2 The Transdichotomous Model

A possible way to make the Random Access Machine more realistic is by observing that large computations can only be done on large machines. Therefore it is reasonable to assume that the size of the machine is related to the size of the problem. This intuition leads us to consider a Random Access Machine where registers and memory cells have a size of $c \log n$ bits, for some constant c, where n denote the size of the instance. The machine does all arithmetic operations between registers in constant time.

This model has been introduced by Fredman and Willard [21, 22], who termed it *Transdichotomous*:

> [...] *because the dichotomy between the machine model and the problem size is crossed in a reasonable manner.* (quoted from [3])

It has been explicitly adopted in late editions of the well-known textbook *Introduction to Algorithms* by Cormen, Leiserson, Rivest and Stein [14]. It seems to us that it is used implicitly in most of the cryptanalytic literature and most algorithmic courses.

2.3 The "Expensive Memory Model"

One of the drawbacks of the two previous computational model is that accessing an unbounded amount of memory takes constant time. Not only is this empirically untrue, but it also violates the known law of physics. The expensive memory model asserts that accessing a large memory has a cost equivalent to $\Theta\left(N^{0.5}\right)$ local operations.

The point is that physically moving data between the memory and processing units requires energy; assuming a mostly flat machine, N bits of memory fit in a disk of radius $\mathcal{O}\left(N^{0.5}\right)$. The underlying assumption is that transporting data over x meters requires an amount of energy proportional to x.

Even at a small scale, memory accesses can be experimentally checked to be more expensive than the evaluation of a simple cryptographic primitive. We just report here the result of a single experiment. We wrote two simple computer programs [2]:

A Iterate $x \leftarrow T[x]$, where T is an array of 2^{24} long integers (64 bits) holding a random permutation. The array requires 128MB.

B Iterate $x \leftarrow \texttt{Speck128/128}_x(0)$ — a constant plaintext block is encrypted with a fresh key at each iteration.

Recall that `Speck128/128` is a presumably secure block-cipher that encrypts 128-bit blocks using a 128-bit key. We used the plain C reference implementation given by its authors [2]. Program B typically runs faster and therefore uses less energy. In other words, evaluating a full-blown block-cipher "costs less" that performing a single memory access that causes a cache miss, for any reasonable notion of cost. Table 1 shows more detailed results. It is well-known that this would only be exacerbated if the size of the data being randomly probed increased, in particular to the point of no longer holding inside a single compute note.

In the RAM model, the complexity (*i.e.* the number of operations) of many attacks is optimized by trading time for memory, eventually reaching $T = M$ in many cases. This is counter-productive in the expensive memory model, as it increases the cost to $M^{1.5}$.

2.4 The AT Model

In the late 70's and early 80's, many lower and upper bounds have been shown on the performance of *flat* VLSI circuits performing common operations: rotation and convolution [41], sorting [10], integer multiplication [12], Fourier transform [37], matrix multiplication [32], ...

[2] They are available at `https://pastebin.com/3D83Kk0s` and `https://pastebin.com/5n6LWprP`.

Program	A	B
Speed (M iterations/s)	11.5	29.5
RAM energy (J)	118.5	40
CPU socket energy (J)	1,053	380

Fig. 1. Performance comparison of two simple programs on a single-core of an Intel Xeon Gold 6130 CPU equipped with 192GB of DDR4 ECC memory at 2666MT/s. Both programs perform 10^8 iterations. The energy readings have been obtained under Linux using the command `perf stat -e power/energy-ram/,power/energy-pkg/ [PROGRAM]`. The "Socket" includes the cores, the full cache hierarchy and the memory controllers. It turns out that the power consumption is comparable in both cases.

These bounds connect the area A of a VLSI circuit and the time T it needs to perform a given computation. They often are of the form $AT \geq n^{1.5}$ or $AT^2 \geq n^2$. The former follows from the latter when $A \geq n$, which is often the case when the input must be entirely memorized by the circuit before the output can be emitted. The point of AT bounds is that a machine of total size A, running for T time steps can perform at most AT operations. These bounds connect the area of a VLSI circuit and the time it needs to perform a given computation. They often are of the form $AT \geq n^{1.5}$ or $AT^2 \geq n^2$. The former follows from the latter when $A \geq n$.

The known AT lower bounds rely on communication complexity arguments, *i.e.* something that is completely ignored by the Random Access Machine model. Figure 2 shows a prime example: a circuit that rotate its n-bit input by $n/2$ bits. This would require $\mathcal{O}(n)$ operations in the RAM model, but is subject to $AT^2 \geq n^2$. Assuming that the signals propagate in constant time along the wiring, the circuit shown by the figure has nearly optimal area $\mathcal{O}(n^2)$. Transporting the data is more costly than suggested by the RAM model.

Using the same kind of arguments, Wiener proved a quite general result:

Theorem 1 ([42], rephrased by us). *In a machine where each of p processors performs uniformly random access to m memory elements at a memory access rate r, the total length of wires is $\Omega\left((pr)^2\right)$. This bound is tight.*

This requires no assumption on the relative locations of processors and memory elements. However this assumes that the computation is in "steady state" for a sufficiently long time. (the memory access rate is the ratio between the number of memory accesses and the total number of instructions executed).

This comes down to saying that a parallel machine where a large number of processors communicate efficiently is larger and costs more than a machine where the same number of processors do not need to communicate at all. The size of the communication network may even dominate the size of the machine. Note that this is a theorem about *machines*, not necessarily *algorithms* and even less *computational problems.*

Based on this result, Wiener argues that attacking double-encryption costs asymptotically more in the AT model than what the number of operations suggests (because it is assumed to need a sophisticated and costly parallel machine).

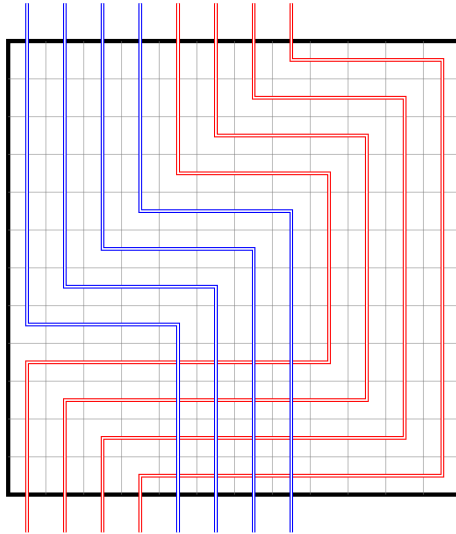

Fig. 2. Possible VLSI circuit that rotates its 8-bit input by 4 bit. Its surface is of order n^2.

Looking again at the case of an algorithm that requires M bits of memory and $\mathcal{O}(M)$ operations, Wiener's theorem suggests to run it on a parallel machine equipped with $p = \sqrt{M}$ processors. This balances the size of memory with the size of the communication network that connects it to the processors. Assuming that the algorithm can be perfectly parallelized, the computation requires $\sqrt{M}$ time steps with a machine of size M, and the AT cost is $M\sqrt{M}$.

The problem is that estimating the cost of a computation in the AT model not only requires us to study an algorithm, but also to study how it could be implemented, mostly in parallel.

In this paper, to avoid problems with the size of the interconnection networks, we consider machines organized as (mostly square) 2D meshes. Each processor has a small memory. It is connected to its north, south, west and east neighbors. The total wire length is proportional to the number of processors. The network link can transmit a finite amount of data during a single time steps. A data packet can only move from one node to the next during a single time step.

2.5 Digression: 2D or 3D?

Both the expensive memory model and the AT model assume *flat* machines. The `NTRU prime` team states that this is reasonable because "*chips are laid out in two dimensions, receiving energy (and dissipating heat) through the third dimension*". These authors conclude that a 3D machine is unrealistic.

Very large computers often span a little in a third dimension. Compute nodes are vertically stacked in racks, themselves often arranged on a 2D grid. The

`K computer` computer (Riken Advanced Institute for Computational Science, Japan), once considered to be the most powerful in the world, had 88 128 compute nodes disposed on a $48 \times 72 \times 24$ physical grid.

In 3D, the expensive memory model means that accessing a memory of size N costs $\mathcal{O}\left(N^{1/3}\right)$ instead of $\mathcal{O}\left(N^{1/2}\right)$. The bound in theorem 1 becomes $(pr)^{3/2}$.

To the best of our knowledge, the *AT* bounds proved by the VLSI community have not been generalized to three dimensions; it is natural to assume that they would yield something like $VT \geq n^{4/3}$ (where V denotes the volume of the machine). Thompson and Kung have shown that a d-dimensional mesh of size n^d can hold and sort n^d small integers in time $\mathcal{O}(n)$, which is consistent with this assumption.

In this paper, we choose to stick with two dimensions, but all the reasoning presenting here could be adapted to three dimensions as well.

2.6 Relations between Models

It is fairly obvious that the number of operations in the RAM model is a *lower bound* on the cost in both the expensive memory model and in the AT model.

Other simple relations are easy to come by. Assume that a program runs in time T using M bits of memory on a Random Access Machine. It costs less than $T\sqrt{M}$ in the expensive memory model: there are at most T memory accesses and each costs less than $\sqrt{M}$. The same computation can be done with cost smaller than $TM^{3/2}$ in the AT model: consider a $2D$ mesh of size $\sqrt{M} \times \sqrt{M}$, where the processor of coordinate $(0,0)$ runs the main program; each other processor contains a single memory cell and act as a storage server; a memory access requires routing the request and the response through the mesh, which requires less than $2\sqrt{M}$ network hops. Thus, the machine has size M and runs in time less than $T\sqrt{M}$.

Both bounds are loose. An interesting example is the classic Hellman Time-Memory trade-off [25]. Its purpose is to efficiently invert a (presumably one-way) function $f : \{0,1\}^n \rightarrow \{0,1\}^n$. Set $N = 2^n$, and recall that the trade-off consists in a preprocessing phase that produces a data structure of size mt, with $mt^2 = N$. Then, inverting the function f on an arbitrary output requires t^2 evaluations of f and t accesses to this data structure. It is customary to set $t = m = N^{1/3}$, which balances time and space. Then, the "online" cost in the expensive memory model matches the number of operations: inverting f requires $N^{2/3}$ operations and $N^{1/3}$ accesses to a memory of size $N^{2/3}$. The cost of these memory accesses is also $N^{2/3}$. Therefore, the technique yields the same cost in the RAM model and in the expensive memory model (and also in the AT model but this is left as an exercise to the reader).

In any case, an access to a memory of size N can be simulated in time $\mathcal{O}\left(\sqrt{N}\right)$ in the AT model, using a machine of size N. This being said, it is difficult to relate costs in the expensive memory and in the AT models. The AT models "charges" the programmer for the memory even it is unused, because the size of the machine includes the size of memory.

Optimizing the cost in the Expensive Memory model can be achieved by reducing the amount of memory needed and/or by reducing the number of memory accesses, as illustrated by the time-memory trade-off discussed above. Optimizing the cost in the AT model means parallelizing the computation and is usually more complex.

The next two sections show separation results. These models may yield costs are incomparable: in section 4 we give an example where the AT cost will be lower than the cost in the expensive memory model, while in section 3 it will be the opposite.

3 The Long Message Attack

The long message attack [28, §9.3.4] is one of the simplest generic attacks there is. It is an interesting example of what can go wrong even in simple cases when working in different computational models. The long message attack applies to naive iterated hash functions that do not use the Merkle-Damgård *strengthening*, namely including the size of the hashed message in the padding of the last block. It has been extended by Kelsey and Schneier to deal with actual Merkle-Damgård hash functions using expandable messages [26]. In this section, we consider the simpler original version.

Consider an ideal (random) compression function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. The Merkle-Damgård hash function H^f iterates f as follows: set an initial internal state $x_0 \leftarrow IV$, split the input in m-bit blocks $M_0, M_1, \dots, M_k$ (pad the last block with a single one bit and sufficiently many zero bits), then compute $x_{i+1} \leftarrow f(x_i, M_i)$. The hash of the input message is simply $H^f(M) = x_{k+1}$. The long message attack forges second preimages for H^f faster than exhaustive search even when f is a random function. The attack is illustrated by fig. 3 and works as follows:

1. Hash the input message M, yielding a sequence of internal states $x_0, \dots, x_{k+1}$. Store them in a static dictionary with constant-time access.
2. Choose a random message block m. Compute $h \leftarrow f(IV, m)$. Probe the dictionary for h: if $f(IV, m) = x_i$, then proceed to the next step. Retry otherwise.
3. A second preimage M' is obtained by replacing the first $i + 1$ blocks of M by m.

To simplify both notations and analysis, let $N = 2^n$ and $k = N^\ell$. Full exhaustive search requires N compression function evaluations, while finding a collision using generic algorithms require only $N^{1/2}$.

Finding the "connecting" message block m such that $f(IV, m) = x_i$ for some internal state x_i requires $N^{1-\ell}$ trials on average. The attack requires $N^\ell + N^{1-\ell}$ compression function evaluations and as many accesses to a memory of size $\mathcal{O}\left(N^\ell\right)$. A static dictionary allowing worst-case constant-time access can be built in linear time [20].

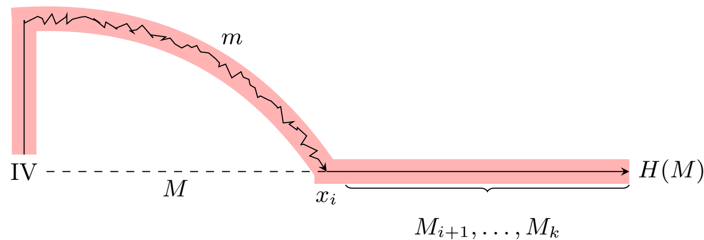

Fig. 3. The long message attack.

3.1 Cost Analysis

In the Random Access Machine model, $\ell = 1/2$ yields an optimal number of operations of $N^{1/2}$ (an operation is, implicitly, a compression function evaluation).

In the expensive memory model, the cost increases because of the (expensive) memory accesses and becomes $N^{\frac{3}{2}\ell} + N^{1-\frac{1}{2}\ell}$. Choosing $\ell = 1/2$ yields a higher yet optimal cost of $N^{3/4}$.

We now show that the cost is necessarily higher in the AT model, and that the optimal target message size is different. A machine must have size N^ℓ to hold the dictionary or even just the input message. However, hashing the input message is an *intrinsically sequential* process. The running time of the machine is thus lower-bounded by N^ℓ, and the AT cost is now lower-bounded by $N^{2\ell}$. It follows that using values of ℓ smaller than 1/2 is mandatory — a fact that apparently evaded previous analyzes in less realistic models of computation.

Assume a 2D mesh of N^ℓ nodes, which is enough to hold the dictionary in a distributed way. We store the dictionary as follows: an n-bit word w is stored by the node of index $w \bmod N^\ell$. Classical results on balls and bins tell us that with high probability the maximum number of values stored by a single node is $\mathcal{O}(n)$. The node of index zero hashes the input message and emits the successive internal states, which are then routed across the mesh towards their destination node.

Once this is done, all nodes repeat the following cycle: generate a random message blocks m, compute $h \leftarrow f(IV, m)$, send the pair (h, m) to the node of index $h \bmod N^\ell$. We are thus facing a parallel mesh routing problem: all nodes simultaneously try to send a message to a randomly chosen target node. Valiant described a simple and efficient algorithm to accomplish this parallel mesh routing in $\widetilde{\mathcal{O}}\left(N^{\ell/2}\right)$ time steps with high probability [38] (the algorithm is simple: route each packet to its correct position along the horizontal axis first, then along the vertical axis). This is easily seen to be optimal using a bisection bandwidth argument: assume that the mesh is split in two halves vertically: on average, half the nodes in the left partition are sending a message to a node in the right partition. Therefore, roughly $0.5N^\ell$ messages must cross the cut during each cycle; the cut consists in $N^{\ell/2}$ network links; therefore moving this amount of data across the cut takes time $\Omega\left(N^{\ell/2}\right)$.

It follows that N^ℓ random blocks can be tested in time $N^{\ell/2}$. This process has to be repeated $N^{1-2\ell}$ times. Processors spend most of their time moving data around, and not evaluating the compression function. This issue of communication complexity is ignored by other models.

The running time of the attack is therefore $N^\ell + N^{1-3\ell/2}$, and the AT cost is $N^{2\ell} + N^{1-\ell/2}$. The optimal choice is $\ell = 2/5$, yielding an AT cost of $N^{\frac{4}{5}}$.

Optimizing for the AT cost is thus a distinct objective than optimizing for the cost in the expensive memory model, and the AT cost can be higher than the expensive memory cost. Here, the difference lies in the fact that the AT models captures problems caused by inherently sequential computations.

The above attack could potentially be improved a little. Instead of waiting that the input message has been completely hashed, all nodes could try to find m (the connecting block) *while the message is being hashed*, even though with a reduced probability of success. We leave it to the reader to check that this only reduces the AT cost by a constant factor.

3.2 The Long Message Attack With Distinguished Points

A well-known technique to reduce both the memory consumption and memory access rate of some algorithms consists in using *distinguished points*. It is the basis of Hellman's Time-Memory trade-off, and it has been used in the late 1980's by Quisquater and Delescaille to find a key-collision on the DES [31]. The technique has then been refined by Van Oorschott and Wiener in the late 1990's [40], and has been used in practice to compute discrete logarithms on elliptic curves, among others.

The technique consists in choosing an easy-to-evaluate predicate π. Bit strings w such that $\pi(w) = 1$ are "distinguished". The key idea is that only distinguished values will be stored in memory. A common way to implement π is distinguish bit strings whose k least significant bits are zero. Choosing the value of k allows us to adjust the proportion of distinguished points. In the sequel, we write N^{-y} the proportion of distinguished points — if w is uniformly random, then $\Pr(\pi(w) = 1) = N^{-y}$. Here is how the modified attack works:

1. Hash the input message M, yielding a sequence of internal states $x_0, x_1, \dots$. Store *only the distinguished ones* in a static dictionary.
2. Choose a random message block m. Compute $h \leftarrow f(IV, m)$. *If h is not distinguished, retry. Otherwise* probe the dictionary for h: if $f(IV, m) = x_i$, then proceed to the next step. Retry otherwise.
3. Assemble a second preimage of M by replacing the first $i + 1$ blocks of M by m.

There are N^{1-y} distinguished points in expectation, and the dictionary contains $N^{\ell-y}$ entries. Therefore the probability that probing the dictionary with a random distinguished point yields a hit is still $N^{\ell-1}$. In this modified attack, the space complexity is reduced to $N^{\ell-y}$, while the time complexity increases to $N^\ell + N^{1-\ell+y}$. This is a pure time-memory trade-off where the search phase is governed by $TM = N$. The dictionary is still probed $N^{1-\ell}$ times.

Let us examine this in the expensive memory model: there are $N^{\ell-y} + N^{1-\ell}$ accesses to a memory of size $N^{\ell-y}$, in addition to $N^{\ell} + N^{1-\ell+y}$ local evaluations of the compression function. The cost is therefore:

$$N^{\frac{3}{2}\ell-\frac{3}{2}y} + N^{1-\frac{\ell}{2}-\frac{y}{2}} + N^{\ell} + N^{1-\ell+y}.$$

Minimizing the exponent means solving a linear program in two variables. The optimal solution is $\ell = 3/5$ and $y = 1/5$, yielding a cost of $N^{\frac{3}{5}}$. This balances the number of local operations and the cost of memory accesses. It also balances the two phases of the attack (hashing M and probing the dictionary). The total amount of memory needed is $N^{\frac{2}{5}}$, and there are this many memory accesses in each phase. This improves upon the basic attack.

Let us now consider the AT model. We consider a 2D mesh of N^p nodes, with $p \geq \ell - y$ (so that the mesh has enough memory to store the dictionary). The mesh is split into square partitions of size $N^{\ell-y}$. Each partition stores a copy of the dictionary. The rate N^{ρ} at which candidate distinguished points are checked against the dictionary obeys two constraints:

- The speed at which processors generate them: $\rho \leq p - y$.
- The network bisection bandwidth: $\rho \leq p/2$.

The wall-clock time needed to produce and test $N^{1-\ell}$ distinguished points is therefore $N^{1-\ell-\rho}$. The AT cost of the whole attack is then $N^{\ell+p} + N^{1-\ell-\rho+p}$.

Minimizing the cost again amount to solving a linear program in ℓ, y, p, ρ. The optimal solution is $\ell = 3/7$, $y = 1/7$, $p = 2/7$ (so the mesh is just large enough to store the dictionary), and the optimal AT cost is $N^{\frac{5}{7}}$.

3.3 Summary and Discussion

Figure 4 summarizes the situation. We end up with five different cost exponents with four different target message lengths. Which one is right? At the very least, this shows that optimizing the cost in the three computational models is a distinct task. Depending on the model, the "best" target message size is different.

Using distinguished points increases the cost in the RAM model but lowers it in both the expensive memory and the AT model. The cost of the long message attack in the AT model is always higher than in the expensive memory model. The ratio of distinguished points and the optimal target message size are always different.

4 HMAC Collisions

This section discusses the problem of evaluating many times a function that has a *large description*. This is a reoccurring pattern in generic attacks (other applications include for instance the enumeration of high-degree boolean polynomials). The point this section makes is that this typically results in higher

Model	Msg. size (x)	y	# f	# Mem. accesses	Mem.	Cost
	Basic Attack					
RAM	1/2	-	1/2	1/2	1/2	1/2
Expensive Mem.	1/2	-	1/2	1/2	1/2	3/4
Area × Time	2/5	-	3/5	3/5	2/5	4/5
	With Distinguished Points					
RAM	1/2	0	1/2	1/2	1/2	1/2
Expensive Mem.	3/5	1/5	3/5	2/5	2/5	3/5
Area × Time	3/7	1/7	5/7	4/7	2/7	5/7

Fig. 4. Cost of the long message attack, with and without distinguished points, in various computational models. The values given are exponents in base 2^n, *i.e.* a cell containing α must be read as $2^{\alpha n}$.

costs in the expensive memory model than in the AT model. This is important because it suggests that the expensive memory model way *overestimate* costs in some situations. This makes the model unsuitable for security analyses, as discussed in section 1.2. The problem we expose is that not all memory accesses are equally costly. In some cases, the cost of random memory accesses to a large memory of size M can be reduced to $\mathcal{O}(1)$, in the AT model.

We illustrate this by considering the following problem: given a (long) message M, find two HMAC keys $k_0 \neq k_1$ such that $\mathsf{HMAC}(k_0, M) = \mathsf{HMAC}(k_1, M)$.

To begin with, we extend the construction of an iterated hash function given in section 3 by writing $H^f(k, M)$ to denote the hash function uses the key k as an initialization vector for the iteration. Then, we define as usual the classic Message Authentication Code:

$$\mathsf{HMAC}(k, M) = H(k \oplus \texttt{opad}, H(k \oplus \texttt{ipad}, M)),$$

where `ipad` and `opad` are two fixed n-bit constants.

Let again N denote 2^n and let N^ℓ be the size of M in message blocks ($\ell < 1/2$). A collision can be found using only a constant amount of additional memory, for instance using Floyd's cycle-finding algorithm. We describe here a solution using the parallel collision search of van Oorschott and Wiener [40], because it will be useful later on. We again assume a proportion of distinguished points equal to N^{-y}, with $y < 1/2$. In order to make this paper self-contained, the well-known algorithm is described below and illustrated by figure 5.

Collision Detection. We search colliding trails. A trail ends at a distinguished point and it is completely described by its initial value and its length. The attack starts with an empty dictionary.

1. Choose a uniformly random initial value $s \in \{0, 1\}^n$. Set $x \leftarrow s$ and $i \leftarrow 0$.
2. While x is not distinguished, do: $x \leftarrow \mathsf{HMAC}(x, M)$ and $i \leftarrow i + 1$.
3. Probe the dictionary for key x. If x is already a key in the dictionary, then we have two trails ending with x. Run the collision location phase.
4. Store the association $x \mapsto (s, i)$ in the dictionary

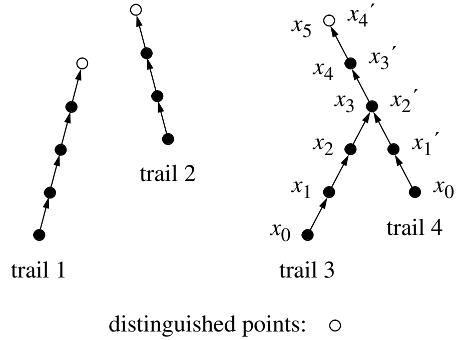

Fig. 5. Collision between trails in the parallel collision search algorithm. Image: [40].

5. Return to step 1.

COLLISION LOCATION. We have two trails (s, i) and (s', i') with a common endpoint. Hopefully, this will lead to the discovery of an actual collision. W.l.o.g. we assume that $i' \geq i$.

1. While $i' < i$, do: $s' \leftarrow \mathsf{HMAC}(s', M)$
2. If $s = s'$, report failure
3. While $s \neq s'$, do:
   (a) $r \leftarrow s$ and $r' \leftarrow s'$
   (b) $s \leftarrow \mathsf{HMAC}(s, M)$
   (c) $s' \leftarrow \mathsf{HMAC}(s', M)$
4. Return the colliding keys r and r'

Van Oorshott and Wiener [40] show that the expected number of HMAC evaluations in the collision detection phase is $N^{1/2}$, while it is of order N^y in the collision location phase. The dictionary stores $N^{\frac{1}{2}-y}$ trails on average when a collision is found.

This requires $N^{1/2}$ evaluations of HMAC, and therefore $N^{\frac{1}{2}+\ell}$ compression functions evaluations in total. This settles the complexity analysis in the RAM model.

4.1 Expensive memory Model

In the expensive memory model, the cost is higher: each evaluation of HMAC must read the whole message M from memory. This requires N^ℓ accesses to a memory of size N^ℓ. These accesses are sequential and thus not random, but the model does not say anything about this. In the collision detection phase, these memory accesses cost $N^{\frac{1}{2}+\frac{3}{2}\ell}$, which dominates the number of local operations.

However, the algorithm can be modified in order to decrease its cost by reorganizing memory accesses. *Blocking* is a well-known technique that improves data locality and decreases the cost of memory accesses (it is abundantly used in scientific computation). We modify the collision detection phase as follows:

Collision Detection with Blocking. We run N^p trails in parallel, where p is a parameter to be chosen later on.

1. For each $0 \leq i \leq N^p$, do: [Start all trails]
   (a) Choose a uniformly random initial value s_i. Set $x_i \leftarrow s_i$
2. For each $0 \leq i \leq N^p$, do: [Advance all trails]
   (a) Check if x_i is distinguished. If so, process as before when a trail is completed and restart a new trail at a fresh random initial value s_i.
   (b) Set $h_i \leftarrow x_i \oplus \texttt{ipad}$ [initiate $\mathsf{HMAC}(x_i, \cdot)$]
3. For all $0 \leq j < N^\ell$, do: [Evaluate HMAC on all trails]
   (a) Fetch the j-th message block M_j from memory
   (b) For each $0 \leq i < N^p$, do:
      i. Set $h_i \leftarrow f(h_i, M_j)$ [advance $\mathsf{HMAC}(x_i, \cdot)$ by one block]
4. For each $0 \leq i < N^p$, do:
   (a) Set $x_i \leftarrow H(x_i \oplus \texttt{opad}, h_i)$ [Finalize $\mathsf{HMAC}(x_i, \cdot)$]
5. Return to step 2

The loop of step 2–5 handles N^p trails in parallel. This loop will therefore be repeated $N^{\frac{1}{2}-p}$ times on average. The total number of HMAC evaluation is therefore still $N^{1/2}$ as in the basic version. However, the memory access pattern has changed. Each message block fetched from memory is amortized over N^p concurrent HMAC computations.

The memory access in each execution of steps 3.a costs $N^{\frac{\ell}{2}}$; each execution of step 3.b costs $N^{\frac{p}{2}}$. The total cost of the algorithm is therefore

$$\underbrace{N^{\frac{1}{2}-p+\frac{3}{2}\ell}}_{\text{step 3.a}} + \underbrace{N^{\frac{1}{2}+\ell+\frac{p}{2}}}_{\text{step 3.b}}$$

Other steps have negligible cost compared to this. Setting $p = \ell/3$ yields a minimal cost of $N^{\frac{1}{2}+\frac{7}{6}\ell}$. Compared to the original presentation, the cost has been divided by $N^{\ell/3}$. Improving this left as an open problem.

4.2 AT Model

We claim that it is possible to implement this algorithm with a *smaller* cost in the AT model using a parallel machine. The main idea is simple: *broadcasting* the content of a memory location to N processors essentially costs $\sqrt{N}$ if the memory has size N. This means that all processors can fetch the same memory location with constant unit cost. The expensive memory model would have given a cost of $N\sqrt{N}$ for the same amount of memory accesses. Anecdotally, this is explicit in the CUDA programming model for GPUs: memory broadcasts are explicitly said to be much more efficient that random uncoordinated memory accesses.

For the sake of completeness, figure 6 shows a machine with area $\approx 4 \times N$ containing N processors connected in a complete binary tree fashion (this is a classic VLSI design). Each processor holds a memory cell. To broadcast the

content of a memory cell to all the other processors, the value is first routed up through the tree to the root (this takes $\sqrt{N}$ time steps). Then it is routed down the tree towards all processors, also in $\sqrt{N}$ steps. The whole process can be pipelined, assuming that network links between processors are full-duplex. This means that a new memory cell can be broadcasted to all processors at each time step.

We now describe a machine that finds HMAC collisions with optimal AT cost. It runs the parallel collision search algorithm using N^ℓ processors. Let $\phi : \{0,1\}^n \to \{0,1\}^\ell$ be the function that returns the ℓ most significant bits.

In the collision detection phase, each processor picks a random key x_0, then computes the sequence $x_{i+1} \leftarrow \mathsf{HMAC}(x_i, M)$. The iteration stops when x_i has its $\frac{n}{2} - \ell$ least significant bits equal to zero. Once all processors have stopped, they move to the collision location phase: each processors sends the message (x_0, x_i, i) to the target processor $\phi(x_i)$ on the mesh. If (at least) a processor receives (at least) two messages, then a collision will be detected almost surely.

We claim that this machine outputs a MAC collision with constant probability. The length of the trails computed by each processor follow a geometric distribution of expected value $N^{\frac{1}{2}-\ell}$, so that the total expected number of evaluations of HMAC is $N^{1/2}$. It follows from the analysis of [40] that the probability of finding a collision is high.

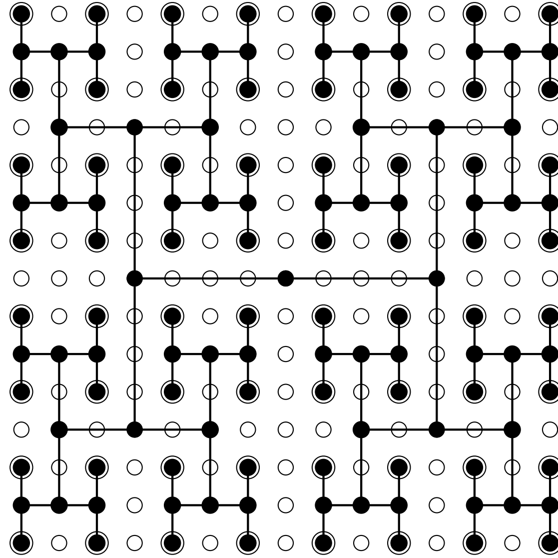

Fig. 6. A mesh of area $\approx 4 \cdot 2^n$ with 2^n (circled) nodes linked in a complete binary tree communication pattern.

The machine has N^ℓ processors arranged in a tree-like fashion as in figure 6. Each processor has a constant number of n-bit memory cells, so the machine is

actually of size $\widetilde{\mathcal{O}}\left(N^\ell\right)$. The input message M is distributed among the memory of all processors: the i-th processor holds the i-th message block.

The machines does $N^{\frac{1}{2}-\ell}$ "cycles" ; during each cycle, each processors computes the next term of its sequence. A cycle requires N^ℓ time steps. During the i-th step of a cycle, the i-th processor emits the i-th message block; it is routed upwards the root of the tree, and the downwards all the other leaf nodes. The process is fully pipelined: a single message block reaches all processors in $\mathcal{O}\left(N^{\frac{\ell}{2}}\right)$ time steps, but a new message block is pushed into the pipeline at each time step. Once the pipeline is full, all processors receive the complete message M every N^ℓ time steps. This allows the N^ℓ processors to evaluate HMAC on N^ℓ distinct keys in N^ℓ time steps. The collision detection phase therefore requires $N^{1/2}$ time steps.

In the collision location phase, we are facing the same network routing problem as before: each processors sends a small packet to another, randomly chosen processor. Valiant's routing algorithm can be used again. Then, once collisions have been detected, they must be located. This requires $N^{1/2}$ time steps. Note that reading the input message M from the outside world takes less time, even if done sequentially.

It follows that this machine solves the problem with AT cost $N^{\frac{1}{2}+\ell}$. Because this matches the number of operations required in the RAM model, we assume that this cannot be improved.

4.3 Summary and Discussion

We exhibited a situation where the expensive memory model overestimates costs compared to the AT model. Calibrating the size of a cryptographic key on the basis of the best attack in the expensive memory model would lead to a potential attack in the AT model, which seems *more* realistic, not less. We conclude that the expensive memory model should be manipulated we care, if not avoided altogether.

5 Generic Key-Recovery Attacks Against Feistel Networks

In this section, we show that reducing the memory complexity of an attack is not automatically a way to reduce its cost in either the expensive memory or the AT models. It can even have an adverse effect.

We illustrate this with key-recovery attacks on Feistel networks. We assume that a block cipher encrypts n-bit blocks using a $2n$-bit master key using a Feistel network. We further assume that round subkeys are $\frac{n}{2}$−bit long, and that their are derived from the $2n$-bit master key in a secure way. This models, for instance, is the CAST-128, DEAL-256, etc. The i-th round transforms the n-bit internal state (L_i, R_i) into $(L_{i+1}, R_{i+1}) = (R_i \oplus F_{k_i}(L_i), L_i)$.

We discuss two key-recovery attack against seven rounds, using 4 plaintext-ciphertext pairs. Both are illustrated by figure 7. All values obtained while encrypting the k-th pair are denoted by $X_i^{(k)}$. The vector of all values $X_i^{(0)}, \ldots, X_i^{(3)}$ is denoted by $X_i^{(\star)}$. We again write $N = 2^n$.

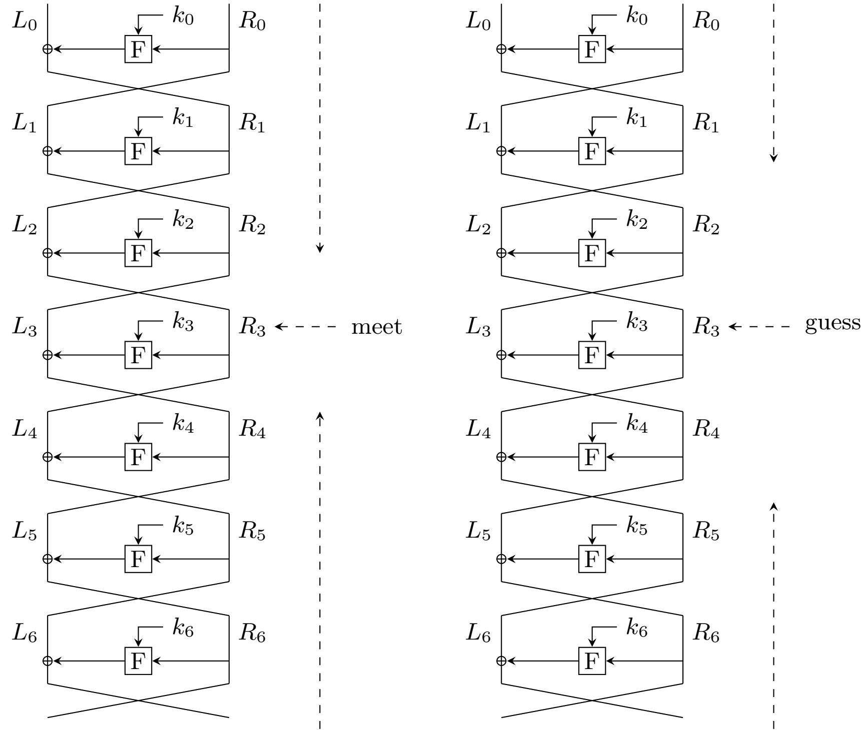

Fig. 7. Meet-in-the-middle attack on 7 Feistel rounds (left) and improved version based on the dissection idea [18] (right).

5.1 Basic Meet-in-the-Middle Attack

A simple meet-in-the middle attack breaks 7 rounds using four known plaintext-ciphertext pairs. It is described by fig. 7. Here is a high-level description.

1. For each value of k_0, k_1, k_2, do:
   a. Partially encrypt the four plaintexts to obtain $R_3^{(\star)}$.
   b. Store the association "$R_3^{(\star)} \mapsto k_i$" in a dictionary.
2. For each value of k_4, k_5, k_6, do:
   a. Partially decrypt the ciphertexts to obtain $R_3^{(\star)}$.

b. Query the dictionary on $R_3^{(\star)}$. This yields candidates for (k_0, k_1, k_2).
c. For each suggested triplet (k_0, k_1, k_2), do:
   i. Enumerate all possible values of k_3 and check the full key against the known plaintext-ciphertext pairs.

The attack requires $N^{1.5}$ units of memory (a "unit" is assumed to hold at least n bits). The number of operations is also of order $N^{1.5}$: the outer loops in steps 1 and 2 do $N^{1.5}$ iterations. Step 2.c.i does $N^{0.5}$ iterations, but it is only expected to be executed N times. In the RAM model, this is faster than exhaustive search on the $2n$-bit master key.

5.2 Improved Attack using Less Memory

Dunkelman, Keller, Dinur and Shamir [18] have improved this basic attack, reducing its memory complexity to N instead of $N^{1.5}$, while leaving its time complexity unchanged at $N^{1.5}$. In the RAM model, this is obviously a strict improvement.

The main idea consists in guessing $R_3^{(0)}$ first. This reduces from $N^{1.5}$ to N the number of triplets (k_0, k_1, k_2) or (k_4, k_5, k_6) that are admissible. The actual attack works as follows.

Offline phase. Initialize an empty dictionary G. Then for each x, k, store the association "$(x, F_k(x)) \mapsto k$" in G. The dictionary stores N key-value pairs. Each key may yield zero, one, or several values.

Online phase. For each $R_3^{(0)}$, do:

1. Initialize an empty dictionary H. For each k_0, k_1 do:
   (a) Partially encrypt the first plaintext to obtain $L_2^{(0)}, R_2^{(0)}$.
   (b) Query G to find all values of k_2 such that $F_{k_2}(R_2) = L_2^{(0)} \oplus R_3^{(0)}$.
   (c) For each suggested k_2, partially encrypt all plaintexts to obtain $R_3^{(\star)}$ and store the association "$R_3^{(1,2,3)} \mapsto k_0, k_1, k_2$" in a dictionary H.
2. For each k_5, k_6, do:
   (a) Partially decrypt the first ciphertext to obtain $R_4^{(0)}, R_5^{(0)}$.
   (b) Query G to find all values of k_4 such that $F_{k_4}(R_4) = R_5^{(0)} \oplus R_3^{(0)}$.
   (c) For each suggested k_4, partially decrypt all ciphertexts to obtain $R_3^{(\star)}$.
   (d) Query the dictionary H on $R^{(1,2,3)}$. This yields candidates for (k_0, k_1, k_2).
   (e) For each suggested (k_0, k_1, k_2) do:
      i. For each k_3, do: check the full key against all plaintext-ciphertext pairs.

The expected number of keys suggested by each probe to G is 1. Therefore H stores N bindings on expectation. The memory complexity is thus reduced to N. Step 2.e.i is executed $N^{1/2}$ times per iteration of the outer loop. Each iteration of the outer loop requires N operations on expectation.

5.3 Modified Basic Attack in the Expensive Memory Model

In the expensive memory model, the basic attack costs $N^{2.25}$ while the improved attack costs N^2. Neither attack is better than exhaustive search when accessing memory is not free. By the way, had we considered 3D machines, then the reduced cost of memory accesses would make the improved attack less costly than exhaustive search. This is the curse of (low) dimensionality.

In this remaining of this section, we show that the *basic* attack can be salvaged, while the *improved* one cannot. The point is that the basic attack is amenable to a time-memory trade-off that *increases* the number of local operations to $N^{1.875}$, including $N^{1.5}$ accesses to a *smaller* memory of size $N^{0.75}$.

It is indeed possible to turn the tide using distinguished points. The basic attack essentially consists in finding all the collisions between two expanding functions from $1.5n$ bits (3 subkeys) to $2n$ bits (4 values of R_3). Say that F_0 maps k_0, k_1, k_2 to the values of R_3, while F_1 maps k_4, k_5, k_6 to the sames values. We expect to find about 2^n collisions $F_0(x) = F_1(x)$ with $x \neq x'$. For each collision between F_0 and F_1, the values of k_3 are searched exhaustively.

The parallel collision search algorithm of van Oorshott and Wiener can again be used to find these collisions efficiently [40]. We briefly recast their ideas applied to the problem of finding all collisions between two expanding functions $F_0, F_1 : \{0,1\}^k \mapsto \{0,1\}^{k+\ell}$. About $2^{k-\ell}$ such collisions are expected. First, we consider modified functions $\overline{F_0}, \overline{F_1}$ with output truncated to k bits; finding all collisions between F_0 and F_1 can be done by finding all the collisions between $\overline{F_0}$ and $\overline{F_1}$ (there are 2^k of them in expectation), and checking each of them against the original functions. This reduces the problem to functions with identical domain and range.

Let us be given a new predicate $P : \{0,1\}^n \rightarrow \{0,1\}$; we define a new function $g_P : x \mapsto F_{P(x)}(x)$. A collision $g_P(x) = g_{P'}(x')$ $(x \neq x')$ reveals a collision between $\overline{F_0}$ and $\overline{F_1}$ if and only if $P(x) \neq P'(x')$. We therefore have a large family of functions $g_\star$; finding many collisions between members of this family reveals a significant fraction of collisions on the original functions.

Finding all collisions on $g_\star$ is done using distinguished points. Let us again assume that a fraction θ of k-bit strings are distinguished. A memory of size M is shared among all processors. Each processor chooses a random starting point x_0 and a random predicate P; it iterates the g_P function, setting $x_{i+1} \leftarrow g_P(x_i)$, until a distinguished point is reached. Then it stores the triplet (x_0, P, i) into the memory cell of index $h(x_i)$, where h is some hash function. If this memory cell already contained a triplet with the same end point, then we may have found a collision. In all cases, the memory cell is overwritten and the process continues.

The heuristic analysis of Van Oorschott and Wiener states that a new collision is found every $\theta 2^k/M + 2/\theta$ iterations of g on average. Finding 2^k collisions thus requires $\theta 2^{2k}/M + 2^k/\theta$ work. Choosing $\theta = \sqrt{M/2^k}$ minimizes this amount. The process then requires about $2^{1.5k}/\sqrt{M}$ operations and 2^k accesses to a memory of size M.

In the Expensive Memory model, this costs $2^{1.5k}/\sqrt{M} + 2^k\sqrt{M}$. The optimal amount of memory is thus $2^{k/2}$, and the total cost is $2^{\frac{5}{4}k}$. There are $2^{1.25k}$ "local"

evaluations of g, as well as 2^k accesses to a memory of size $2^{k/2}$. Wiener shows that this cost can be matched in the AT model [42].

Back to the basic meet-in-the-middle attack on Feistel Networks, this means that all collisions between F_0 and F_1 can be found with cost $N^{1.875}$, barely less than exhaustive search. Performing the N exhaustive searches on k_3 costs less. The attack is therefore still viable in the expensive memory model, but only using distinguished points.

5.4 Improved attack in the Expensive Memory Model

The improved attack repeat $N^{1/2}$ times a procedure that searches all collisions between two expanding functions $F_0, F_1 : \{0,1\}^n \rightarrow \{0,1\}^{1.5n}$. The problem is that these functions now have an exponentially large description because they access the dictionary G, which is of size N. It is therefore not possible to evaluate them "locally" without accessing a large memory, and the whole parallel collision search approach seemingly breaks down. Indeed, the whole point of using distinguished points is to reduce the space complexity below N.

Given M memory cells, choosing the optimal proportion of distinguished points requires $\sqrt{N^3/M}$ evaluations of F_0 and F_1, with $M \leq N$. To beat exhaustive search, we would need a way to evaluate the functions with cost strictly less than $N^{0.5}$. We are not aware of any such technique. Using the offline phase proposed in [18] and described above, evaluating the functions costs exactly $N^{0.5}$ (one access to a memory of size N). The size of the data structure computed during the offline phase could be reduced using Hellman's Time-Memory tradeoff $N^{0.5}$ times for each value of k. This would yield a data structure of size $N^{\frac{5}{6}}$ and evaluating the functions would require $N^{\frac{1}{3}}$ local operations plus $N^{\frac{1}{6}}$ accesses to the data structure. The cost of these memory accesses is $N^{\frac{7}{12}}$, which is even worse than before.

5.5 Summary and Discussion

The basic attack could be modified to cost $N^{1.875}$ in the expensive memory model. On the other hand, we fall short of finding a way to make the improved attack less costly than exhaustive search in the expensive memory model. This comes at a surprise, because the improved attack uses less memory, and as many operations as the basic one.

This shows that optimizing for more sophisticated costs models goes beyond reducing the space complexity. It may require to make the attacks *worse* in the more common RAM model of computation.

6 Generic Key-Recovery Attacks Against HMAC

We conclude this paper with the study of a sophisticated generic attack. We show that it is more costly than brute force in the expensive memory model.

Attack	# Operations	# Mem. Access	Mem. Size	EM Cost
Basic	1.5	1.5	1.5	2.25
Improved [18]	1.5	1.5	1	2
Modified Basic	1.875	1.5	0.75	1.875

Fig. 8. Cost of the key-recovery attacks on 7-round Feistel networks. The values given are exponents in base 2^n, *i.e.* a cell containing α must be read as $2^{\alpha n}$.

We then show that it can be modified to beat brute force in this same model. Lastly, we show that it is always worse than brute force in the AT model.

The universal forgery attack against HMAC presented by Peyrin-Wang [30], and later improved by Guo-Peyrin-Sasaki-Wang [24], is one of the most spectacular generic attack of the last decade. We consider the setting where the internal state of the hash function has the same size as the tag, namely n bits. The attack even works against NMAC:

$$\mathsf{NMAC}(k_1, k_2, M) = H(k_1, H(k_2, M)),$$

The attack forges a valid tag for any message M by computing a second preimage $M' \neq M$ for the inner hash function: $H(k_2, M) = H(k_2, M')$. This is accomplished without knowledge of k_2. Then, M' can be submitted to the MAC oracle, and the resulting tag is also valid for M. In the sequel, we again use the notation $N = 2^n$ and we write N^ℓ the size of the target message M.

All known second-preimage attacks are variants of the long message attack discussed in section 3, and they require access to the sequence of internal states $x_0, x_1, x_2, \ldots$ obtained while hashing the input message M. Here, the adversary faces a major obstacle because these internal states are not available: the hashing process starts from k_2, which is unknown. Identifying a single internal state is sufficient to recover all the next ones and make existing second-preimage attacks work. Once this is done, computing the second preimage (using the Kelsey-Schneier attack) requires $N^{1-\ell}$ compression function evaluations.

To recover one internal state value, the attacks exploits the functional graph of the function $g : \{0,1\}^n \rightarrow \{0,1\}^n$ defined by $g(x) = f(x, m)$, where m is a fixed arbitrary message block. This graph has N nodes and there is a directed edge $x \rightarrow y$ if and only if $y = g(x)$. Properties of these graphs for random functions g have been extensively studied [19]. Each connected component of the graph contains a single cycle into which several trees are grafted. The *height* of a node is zero if it belongs to a cycle, otherwise it is its distance to the cycle of its component.

A clever procedure (not described here) enables the adversary to discover the heights of all internal state values x_i's in the functional graph of g. Because the x_i's are random and g behaves as a random function, the heights of the x_i's are expected to be less than $N^{\frac{1}{2}}$; if the message size obeys $\ell \leq \frac{1}{4}$, we expect the heights to be all distinct. The two successive versions of the attack differ in how they exploit this information.

6.1 First Version (Peyrin-Wang) [30]

This version of the attack targets messages of size $\ell = \frac{1}{6}$. Its time and space complexity are both $N^{\frac{5}{6}}$.

The attack requires the construction of "filters" for all internal state values x_i, namely of a pair of message blocks $m_i \neq m_i'$ such that $f(x_i, m_i) = f(x_i, m_i')$. This enables an efficient decision procedure to test if x_i is equal to a candidate value $\hat{x}$: we just have to test if $f(\hat{x}, m_i) = f(\hat{x}, m_i')$. There are no false negatives and the probability of false positive is extremely small. The filters can be obtained by finding MAC collisions: For all $i \leq N^\ell$, the adversary searches a pair of blocks $m_i \neq m_i'$ such that the tags of $M_0 \| \dots \| M_i \| m$ and $M_0 \| \dots \| M_i \| m'$ are equal. This requires $N^{\ell + \frac{1}{2}}$ queries to the oracle with messages of size less than N^ℓ. Following the authors of the attack, we assume that submitting a x-block query to the MAC oracle requires x "elementary operations". All-in-all, building the filters takes time $N^{2\ell + \frac{1}{2}}$.

To identify one of the x_i, the attacker maintains a set Y of $N^{1-\ell}$ bit strings. With constant probability, at least one of the x_i's belongs to Y. The attack works as follows: for each target internal state x_i, let h be its height; for each $\hat{x} \in Y$ of height h, use the i-th filter to test if $x_i = \hat{x}$ in constant time. Because the heights of the x_i's are all distinct, each element of Y is tested at most once, and this takes time at most $N^{1-\ell}$.

It remains to see how the data structure holding Y can be built. This can be done in an offline phase and reused for several input messages. The following procedure is suggested to accumulate N^y nodes with $y \geq 1/2$. Until enough nodes have accumulated, pick a random bit string u_0 then iterate $u_{i+1} = g(u_i)$ until a collision is detected ($u_i = u_j$ with $i < j$) or u_i already belongs to Y. Once a chain ends due to either stopping criteria, add all the encountered nodes to Y. This procedure takes N^y units of time; the data structure uses N^y units of space.

Choosing $\ell = 1/6$ balances the number of operations required to build the filters, build Y, identify an internal state value and compute the second preimage.

In the Expensive Memory Model. The attack uses $N^{\frac{5}{6}}$ time and $N^{\frac{5}{6}}$ space. In the expensive memory model, just reading each memory cell once costs at least $N^{5/4}$ and the attack is doomed. Adjusting the parameter ℓ does not help: because the data structure that contains Y has to be read entirely, we would need to use values of ℓ greater than $1/3$ to beat brute force. Not only is this forbidden by the description of the attack (arguments to establish its complexity require $\ell \leq 1/4$), but it would drive the cost of creating the "filters" to at least $N^{7/6}$.

Fortunately, the space complexity of the data structure holding Y can be reduced. This can again be achieved using distinguished points, as observed by Dinur [17]. Assume that we want to represent N^y nodes with $y \geq 1/2$. The key idea is to consider that a fraction N^{y-1} of the nodes is distinguished and to only store distinguished points. The compacted data structure requires N^{2y-1} space. Building it now works as follows: start each chain at a random distinguished

point x_0 ; compute the iterates $x_{i+1} = g(x_i)$; if x_i is distinguished, then check if it belongs to Y ; if so, stop and begin a new chain ; otherwise, add it to Y and continue iterating. A minor extension of this procedure also stores the height of each distinguished point. The time complexity of the construction is unaltered. Building the data structure requires N^y local operations and N^{2y-1} accesses to a memory of size N^{2y-1}. The cost of memory accesses dominates the cost of local operations when $y \geq 3/4$.

When it is used in attacks against MACs with $y = 1 - \ell$ and $\ell \leq 1/4$, building the data structure thus costs $N^{\frac{3}{2}(1-2\ell)}$. Its size is reduced from $N^{1-\ell}$ originally to $N^{1-2\ell}$. The problem is that it is no longer possible to iterate over nodes of a given height. However, it is possible to efficiently iterate over all nodes while knowing the height of the current node (this is easy if the height of all distinguished points is known). The attack can then be modified as follows: for each node $\hat{x}$ in Y, let h be its height, and let x_i denote the single value of height h in the sequence of input internal states (if there is any). Use the procedure described above to test if $x_i = \hat{x}$. The number of operations is unchanged. The online search phase now requires $N^{1-2\ell}$ accesses to the data structure of size $N^{1-2\ell}$ and $N^{1-\ell}$ access to an associative array of size N^{ℓ} that maps a height h to a pair (x_i, m_i, m'_i). The cost of the online search phase is therefore $N^{\frac{3}{2}(1-2\ell)} + N^{1-\ell/2}$. Note that this dominates the cost of both building the data structure and finding the second preimage naively.

Creating the filters is done "online" and requires interactions with the MAC oracle. What is the cost of these interactions? In fact, what is the actual interaction mechanism? In the classic textbook [23], Goldreich suggests to use oracle Turing machines: a special oracle tapes receives the queries to the oracle; upon invocation, it is erased and replaced with the answer. This implies that invoking the oracle on a long message takes time proportional to the length of the message. But it also implies that the message has to be read first, which in our case entails $N^{\frac{1}{2}+2\ell}$ accesses to a memory of size N^{ℓ}. The cost of sending these queries to the oracle is therefore $N^{\frac{1}{2}+\frac{5}{2}\ell}$.

The total cost, including the creation of the filters, is therefore

$$N^{\frac{1}{2}+\frac{5}{2}\ell} + N^{\frac{3}{2}(1-2\ell)} + N^{1-\ell/2}.$$

This reaches a minimum of $N^{21/22}$ with $\ell = 2/11$. This is still better than exhaustive search... but barely. The precise mechanism of interaction with the oracle could change the whole complexity of the attack.

6.2 Second Version (Guo-Peyrin-Sasaki-Wang) [24]

This improved version of the attack changes two aspects. First, the "filters" are improved. Suppose that the adversary has obtained a MAC collision between $M \| m$ and $M \| m'$ where $m \neq m'$ are two message blocks. Finding this MAC collision requires $N^{\frac{1}{2}}$ queries to the MAC oracle with messages of size N^{ℓ} (this is less than in the first version). Obtaining this single "super-filter" requires $N^{\frac{1}{2}}$ MAC queries and $N^{\frac{1}{2}+\ell}$ operations.

Then, given a candidate value $\hat{x}$ and a position i, testing if $x_i = \hat{x}$ can be done without access to the oracle by checking whether the following equality holds:

$$f(f(\dots f(f(x_i, M_i), M_{i+1}) \dots, M_{N^\ell}), m) = f(f(\dots f(f(\hat{x}, M_i), M_{i+1}) \dots, M_{N^\ell}), m').$$

Performing this test requires N^ℓ operations (this is more than in the first version).

The second improvement comes from the (heuristic) observation that the heights of elements of Y are approximately *uniformly distributed* integers less than $N^{\frac{1}{2}}$. This implies that Y contains $N^{\frac{1}{2}-\ell}$ nodes of each height. In the online search phase, for each internal state x_i, the number of tested candidates is approximately $N^{\frac{1}{2}-\ell}$; each test requires less than N^ℓ operations. The total number of operations of the search phase is therefore $N^{\frac{1}{2}+\ell}$.

Choosing $\ell = 1/4$ minimizes the total number of operations, which is reduced to $N^{\frac{3}{4}}$. It must be noted that only a small fraction of the data structure is actually read during the online phase of the attack, namely $N^{\frac{1}{2}}$ nodes out of $N^{\frac{3}{4}}$.

In the Expensive Memory Model Finding the "super-filter" costs $N^{\frac{1}{2}+\frac{3}{2}\ell}$: each query to the MAC requires N^ℓ accesses to a memory of size N^ℓ containing the input message.

Testing if $x_i = \hat{x}$ requires N^ℓ local operations, in addition to N^ℓ accesses to a memory of size N^ℓ. Therefore it costs $N^{\frac{3}{2}\ell}$.

However, a new problem arises: the attack specifically *requires* us to iterate over nodes of Y of a given height. This does not seem compatible with the use of the compact representation of Y (at least, we do not know how to do it). Using the original data structure for Y is impossible (the cost is too high). Using the compact representation, we can only iterate over all nodes of Y, as opposed to those with a specific height. The attack could work as follows. For each $\hat{x} \in Y$: obtain its height h ; access a memory of size N^ℓ to check if there is an intermediate hash value of height h ; if so, perform the test as in the original presentation of the attack.

This tests $N^{\frac{1}{2}}$ candidates. Enumerating all $\hat{x}$ requires $N^{1-\ell}$ local operations and $N^{1-2\ell}$ accesses to a memory of size $N^{1-2\ell}$ (holding Y). It also requires $N^{1-\ell}$ accesses to a memory of size N^ℓ to check if there is a chaining value of height h.

The total cost of this modified version of the attack is then:

$$N^{\frac{1}{2}+\frac{3}{2}\ell} + N^{\frac{3}{2}(1-2\ell)} + N^{1-\frac{1}{2}\ell}$$

This reaches a minimum of $N^{\frac{7}{8}}$ with $\ell = \frac{1}{4}$. The optimal target message size is the same as in the RAM model.

6.3 In the AT Model

A potential problem arises in the AT model with the construction of the data structure representing Y. Using distinguished points, the amount of memory

(and thus the area of the machine) can be reduced to $A \geq N^{2y-1}$ as we have seen earlier, if Y is to represent N^y nodes.

To obtain the heights of nodes in Y, we need to find at least one cycle in the functional graph of g. Nodes on a cycle represent a small fraction $N^{-\frac{1}{2}}$ of the total, but they are relatively easy to find: it suffices to iterate g from an arbitrary point. This can even be done using only a constant amount of memory using classical cycle-finding algorithms. But it requires $N^{\frac{1}{2}}$ *sequential* steps — we do not see how this could be parallelized.

In fact, the problem is even more general: the i-th chain computed by the sequential procedure that builds Y has length $N^{\frac{1}{2}}/\sqrt{i}$. There are N^{2y-1} chains in total, and they could be computed in parallel. However, the machine cannot stop until the longest chains have completed. This implies that the wall-clock running time of a machine that computes Y is lower-bounded by $T \geq N^{\frac{1}{2}}$.

It follows that $AT \geq N^y + N^{2y-\frac{1}{2}}$. This is more than what we had found in the expensive memory model. In the attacks against MACs, this translates to an increased cost of $N^{\frac{3}{2}-2\ell}$ to build the data structure holding Y. Because ℓ is capped at $1/4$, we find that building the data structure is always as costly as exhaustive search.

Both versions of the attack are thus doomed in the AT model for this simple reason.

6.4 Summary and Discussion

This section shows that interesting and sophisticated generic attacks can be valid in a given computational model and invalid in another, slightly more realistic one. The reasoning presented here does not rule out the possibility of a "better than brute force" generic universal forgery attack against HMAC in the AT model, using the same ideas as the attacks of [30, 24]. But at the very least, some major modifications seem to be required (such as dropping the $\ell \leq 1/4$ constraint).

We also remark that it has never been clear how a parallel machine could access an outside oracle. Is the machine allowed to query the oracle in parallel? Is this restricted to processing elements on the surface of the machine or can even interior nodes query the oracle? What is the wall-clock time taken by these interactions? What is the processing power of the oracle? In fact, the attack scenario itself already makes it difficult to consider "realistic" attack machines.

7 Conclusion

In light of the problems discussed in section 4, we claim that

Position 6 *If one wishes to use a more realistic than the RAM model, then it is safest to jump all the way to the AT model, just to be on the safe side.*

References

1. Jean-Philippe Aumasson. Too much crypto. *IACR Cryptol. ePrint Arch.*, 2019:1492, 2019.
2. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. `https://ia.cr/2013/404`.
3. David Benoit, Erik D. Demaine, J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.
4. Daniel J. Bernstein. Circuits for integer factorization: a proposal, 2001. URL: `http://cr.yp.to/papers.html`.
5. Daniel J. Bernstein. Understanding brute force, 2005. Available online: `http://cr.yp.to/papers.html#bruteforce`.
6. Daniel J. Bernstein. Better price-performance ratios for generalized birthday attacks. In *SHARCS'07: Special-purpose Hardware for Attacking Cryptographic Systems*, 2007.
7. Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wen Wang. Classic mceliece, 2017.
8. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. Ntru prime, 2017. Submission to NIST post-quantum call for proposals.
9. Daniel J. Bernstein and Tanja Lange. Batch NFS. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 38–58. Springer, 2014.
10. Gianfranco Bilardi and Franco P. Preparata. Area-time lower-bound techniques with applications to sorting. *Algorithmica*, 1(1):65–91, 1986.
11. Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2020.
12. R. P. Brent and H. T. Kung. The chip complexity of binary arithmetic. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 190–200, New York, NY, USA, 1980. ACM.
13. Stephen A. Cook and Robert A. Reckhow. Time bounded random access machines. *J. Comput. Syst. Sci.*, 7(4):354–375, 1973.
14. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.
15. M. Delcourt, T. Kleinjung, A.K. Lenstra, S. Nath, D. Page, and N. Smart. Using the cloud to determine key strengths – triennial update. Cryptology ePrint Archive, Report 2018/1221, 2018. `https://eprint.iacr.org/2018/1221`.
16. Whitfield Diffie and Martin E. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, 1977.

17. Itai Dinur. New attacks on the concatenation and XOR hash combiners. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 484–508. Springer, 2016.
18. Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. New attacks on feistel structures with improved memory complexities. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 433–454. Springer, 2015.
19. Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354. Springer, 1989.
20. Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $\mathcal{O}(1)$ worst case access time. *J. ACM*, 31(3):538–544, June 1984.
21. Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.
22. Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.
23. Oded Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques.* Cambridge University Press, 2001.
24. Jian Guo, Thomas Peyrin, Yu Sasaki, and Lei Wang. Updates on generic attacks against HMAC and NMAC. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 131–148. Springer, 2014.
25. Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980.
26. John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.
27. Thorsten Kleinjung, Arjen K. Lenstra, Dan Page, and Nigel P. Smart. Using the cloud to determine key strengths. In Steven Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012*, pages 17–39, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
28. Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
29. Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.
30. Thomas Peyrin and Lei Wang. Generic universal forgery attack on iterative hash-based macs. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May*

11-15, 2014. Proceedings, volume 8441 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2014.
31. Jean-Jacques Quisquater and Jean-Paul Delescaille. How easy is collision search? application to des. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 429–434, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.
32. John E. Savage. Area—time tradeoffs for matrix multiplication and related problems in vlsi models. *Journal of Computer and System Sciences*, 22(2):230 – 242, 1981.
33. Adi Shamir. Factoring Numbers in $\mathcal{O}(\log n)$ Arithmetic Steps. *Inf. Process. Lett.*, 8(1):28–31, 1979.
34. John C. Shepherdson and Howard E. Sturgis. Computability of recursive functions. *J. ACM*, 10(2):217–255, 1963.
35. Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 570–596, Cham, 2017. Springer International Publishing.
36. The Rainbow team. Response to recent paper by ward beullens, 2021.
37. C. D. Thompson. Area-time complexity for vlsi. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 81–88, New York, NY, USA, 1979. ACM.
38. L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 263–277, New York, NY, USA, 1981. ACM.
39. Peter van Emde Boas. Machine models and simulation. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 1–66. Elsevier and MIT Press, 1990.
40. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.
41. J. Vuillemin. A combinatorial limit to the computing power of vlsi circuits. *IEEE Trans. Comput.*, 32(3):294–300, March 1983.
42. Michael J. Wiener. The Full Cost of Cryptanalytic Attacks. *J. Cryptology*, 17(2):105–124, 2004. https://doi.org/10.1007/s00145-003-0213-5.