

Robert D. Silverman
RSA Laboratories

Exposing the Mythical MIPS Year

MIPS can be a useful measure of processor speed, and MIPS years can be a useful measure of work. But the MIPS year has been classically misapplied and misused as a measurement of the amount of effort needed to break and compare cryptographic keys.



MIPS (million instructions per second) has been the most popular measure for computer speed since its inception, and people use it as the primary basis for comparing the speed of different computers. But it should come with a set of rules. As long as you use it as a relative, rather than absolute, measure, and as long as you make the comparisons by running similar types of programs on similar types of architectures, MIPS is a useful measure. Historically, however, people have used MIPS without following these rules.

As computers have evolved, the size of problems that can be solved has grown enormously. During the 1980s, the widespread increase in the size, number, and types of computers available allowed scientists to do what earlier had been impossible: combine large networks of heterogeneous computers to solve a single problem. Measuring the overall speed of such a collection of machines became a challenge. The concept of a MIPS year was invented as a way to measure the total effort expended on very large scale problems—specifically the effort needed to break cryptographic keys.

A MIPS year is the number of instructions a 1-MIPS machine can execute in one year. There are about 31 million seconds in a year, so 1 MIPS year represents the execution of about 3.1×10^{13} instructions. Even today, few problems require that much effort, so the term MIPS year is used almost solely within cryptography. The others are all considered “Grand Challenge” problems, such as the calculation of the

mass of a proton from basic principles. Indeed, MIPS year is a term used almost solely within the field of cryptography.

However, as I hope to convince you, the use of MIPS year as a measurement of the amount of effort needed to break and compare cryptographic keys has led to some very inaccurate estimates.

The MIPS year measurement has four basic problems

- Not all instructions are equal, so how do you really know the MIPS rating of a machine?
- Not all programs use the same instructions, so different programs can give different measures of speed even for the same machine.
- The way you typically measure MIPS years expended on a single problem by many computers is to measure the amount of time spent on each machine, multiply that time by the MIPS rating of that machine, and add the individual results. This introduces two possible sources of error: The estimate of machine speed may be inaccurate and the measured time may be inaccurate. A machine may run for, say, two weeks, but it may not have run 100 percent of the time on the problem being studied.
- If you consider *only* the number of instructions executed, you cannot accurately measure the difficulty of hard problems, like breaking the Rivest-Shamir-Adleman (RSA) public-key encryption algorithm, because some algorithms also require

massive amounts of space—both disk storage and RAM. People often ask me how long it will take to break RSA keys of differing size so they can compare those keys with the key sizes required by other cryptographic algorithms. However such a question assumes that *time* is the only computer resource that needs to be compared, an assumption that is not true.

THE ELUSIVE MIPS RATING

Even on modern RISC processors, not all instructions execute in a single cycle. Someone once suggested encasing the DEC VAX/780 in glass and labeling it “1 MIPS.” However, if you actually performed a series of benchmarks based on diverse applications, even this classic 1-MIPS machine would be something closer to a 0.9-MIPS machine. This represents an average speed, taken across many diverse applications.

However, some applications can make a machine appear many times faster, relative to other applications if speed is measured purely by instructions per second. Furthermore if you consider a modern machine rated at, say, 100 MIPS, you would rapidly find that it does not run most applications 100 times faster than a VAX.

Not all instructions are equal

The VAX was perhaps the ultimate complex instruction set computer (CISC). Execution times for its instructions varied by more than three orders of magnitude. Consider the instructions in Table 1. If a program does mostly register-to-register addition, the VAX becomes a 2.5-MIPS machine. On the other hand, if the program has lots of trigonometric evaluations and thus uses the polynomial evaluation (POLYH) instructions heavily, the VAX would become a 0.0017-MIPS machine. It is also somewhat strange that it takes so much longer to move a word from register to memory (MOV Long reg, mem) than in the other direction (MOV Long mem, reg).

What about the machine and compiler?

The use of MIPS years to measure work is particularly dangerous when analyzing the performance of an algorithm. Typically, such an analysis equates time with arithmetic operations. The problem comes when you look for a one-to-one correspondence between arithmetic operations and computer cycles. There isn't one. With pipelining and superscalar execution, a processor often executes more than one operation per cycle. Some instructions interfere with this, while others can have a one- or two-cycle latency before you can use the result of an operation.

Consider the code fragment

$$x[i] = y[i] + z,$$

Table 1. Actual times for instructions executed on a VAX 780 with a floating-point accelerator.

Instruction	Time (μ s)
ADD Long reg, reg	0.40
ADD Long mem, mem	1.72
MUL Long2 reg, reg	6.43
MOV Long mem, reg	0.84
MOV Long reg, mem	1.31
POLYH (4th degree; floating H format)	579.68
ADD Packed reg, reg	18.10

where x , y , and z are integer variables. A good compiler might encode it as one VAX instruction; an average compiler might yield four instructions. As one instruction, it would take 1.7 μ s to execute, making the VAX a 0.58-MIPS machine. As four instructions, it would take 3.39 μ s, which would make the VAX a 1.18-MIPS machine.

So which is it? From an algorithmic view, the work done in each case is the same. Not only that, but the 1.18-MIPS machine is actually slower, although it has a higher MIPS rating.

Counting the number of instructions or time to execute a piece of code depends so heavily on the compiler and machine design that MIPS ratings could vary by a factor of 10. Compilers, depending on how good they are, can rearrange the instruction stream to avoid pipeline stalls, resulting in greater speed without changing any instructions. On a pipelined and superscalar machine, several lines might execute at once. A multiply and add might be two instructions or one. If statements might be done in parallel on a machine with speculative branch execution. The time to execute some code can also depend on whether variables are in the register or the cache. In contrast, for most algorithms, *the number of arithmetic operations does not vary and is well defined.*

Key-breaking algorithms

The elusiveness of the MIPS rating is particularly important in the context of key-breaking algorithms, especially when you use it to determine comparative key sizes for different cryptographic algorithms.

Two classes of public-key cryptographic algorithms are currently in use today. The RSA algorithm depends (in the current state of knowledge) on the difficulty of factoring very large integers. The best algorithms for factoring large numbers are sieve-based algorithms, including the General Number Field Sieve (GNFS) and the Multiple Polynomial Quadratic Sieve (MPQS). These algorithms factor many smaller numbers using a *sieve*—an algorithm for rapidly identifying which among a large set of numbers are divisible by small primes.

The second class of algorithms, of which the Digital Signature Algorithm (DSA) is an example, is based on what is known as the *discrete logarithm* problem. There are two classes of the discrete logarithm problem. The first can be solved using sieve-based algo-

The speed of a computer is very dependent not only on the speed of the processor, but also on the size of the problem being solved.

rithms: Given integers y and g and a prime number p , find x so that $y = g^x \bmod p$. The second class, which is based on the use of *elliptic curves*, cannot (in the current state of knowledge) be solved using sieve-based algorithms. Instead, breaking this second class relies on very fast arithmetic on multiprecision integers, which in turn requires very fast multiplication and division instructions. Most computers today do not do these very quickly because most applications do not need multiplication and division frequently, so it is not worthwhile for CPU manufacturers to devote a lot of hardware to them.

Established benchmarks

Established benchmarks suites, such as Whetstone, Dhrystone, and SPEC (<http://www.nosc.mil>), do not use multiplication and division very much. So if you use one of these as a measure of speed and then try to apply that measure to a key-breaking algorithm, the results can be very misleading. Further, if you look at MIPS measurements given by even these standard benchmarks, you can find contradictory results. Sometimes machine A will appear to be 200 MIPS and machine B to be 100 MIPS for one benchmark, while another benchmark can reverse these numbers.

Part of the problem in assigning MIPS ratings to machines lies in the design of established benchmarks such as Dhrystone. The problem is that the data sets associated with the computations usually don't take up much space. Therefore they usually fit in whatever cache is available, and the computations can run at the machine's full speed.

The one notable exception is the Nsieve benchmark, a sieve-based method. This benchmark is run twice: once with a small data set (8,191 bytes) and once with a large data set (2.5 Mbytes). It is not unusual to see an 18:1 difference in the MIPS ratings of the two benchmarks *on the same machine*, although 3:1 or 4:1 is more typical. Indeed, a Pentium-based PC saw only a 1.7:1 drop-off from the large to small data set. Therefore the speed of a computer is very dependent not only on the speed of the processor, but also on the *size of the problem being solved*.

In contrast, factoring and discrete logarithm algorithms rely much more heavily on integer arithmetic instructions, especially in terms of integer multiplication and division. Sieve-based factoring algorithms rely much more heavily on fast bidirectional memory to processor data transfer than on processing speed. As a result, these methods run at the speed of memory, rather than at the speed of the processor. Under such circumstances, trying to judge how long it will take to break a key based on MIPS estimates of processor speed will be very wrong.

THE ROLE OF MACHINE ARCHITECTURE

A machine's performance depends on more than just its available instruction set. The system bus, memory management unit, primary and secondary data caches, and memory type all affect performance. You just can't give a MIPS rating to a processor in a vacuum. Its performance depends too much on the platform in which it is embedded. Users also sometimes compare the speed of computers based on the clock speed of the processor. But this too can be misleading because quite often machines with a higher clock speed can be slower than those with a lower clock speed because of the machines' architecture.

Caches and memory

Sieve-based algorithms rely almost exclusively on the machine's ability to perform a basic series of operations: select a memory location, fetch a byte from that location, add a value to it, put it back, and update the memory location. Thus, the presence of a cache and the size of that cache greatly influence performance. Retrieving a byte from memory that is not cache-resident can cost two to three cycles of wait states. I have seen a very slow Sparcstation with a large cache outperform much faster machines with smaller (or no) caches. When processing sieve algorithms, a Pentium II running at 233 MHz gives three times the performance of a Pentium at 166 MHz simply because the Pentium II has a large data cache. Yet other benchmarks give MIPS ratings almost in direct proportion to these machines' clock rates.

A large primary or secondary data cache or a low-latency memory can improve performance more than tenfold over otherwise identical processors. With insufficient memory, parameter selection can be less than optimal, resulting in lower performance than on an identical machine with more memory. Yet the MIPS ratings for the two machines would be the same.

Effect of 'special' instructions

Typical benchmarks assume the machine has a basic underlying instruction set. If the machine has additional instructions or is missing others, performance can significantly change. For example, Thinking Machines' CM-5 was a parallel machine based on Sparc processors, but each node also had an accelerator with special instructions useful to cryptography such as population count and index of most and least significant bits. The machine executed some key-breaking algorithms much faster than a machine that was considered its MIPS equivalent.

Suppose a symmetric key algorithm relies heavily on data-dependent rotations. The presence of a rotation instruction can quadruple speed, yet it does not change the machine's MIPS rating. You can have identical 10-MIPS machines, but the one that has a rota-

Debunking MIPS Year Claims

RSA Laboratories has published a set of challenge numbers for factoring algorithms. These are large integers that are the product of two nearly equal primes, which are the hardest kind of integer to factor. They are designated by their size: RSA-129, for example, is an integer with 129 decimal digits. Several challenge numbers have been broken via large distributed efforts over the Internet using very heterogeneous sets of machines.

It was reported that RSA-129 was broken with an effort of 5,000 MIPS years via the Quadratic Sieve. However, this number might be viewed with some suspicion. The effort involved a very large heterogeneous set of machines distrib-

uted across the Internet. We only have rough knowledge of the exact makeup of this set, how long each machine ran, and what fraction of time it was working on the problem. Thus, not only are the estimates of machine MIPS in doubt, but there is a fair amount of potential error in estimating how long each machine ran.

It was reported that RSA-130 was broken in about one-third that time via the Number Field Sieve, but that it could have been done in one-tenth the time if the sieving machines had had more memory. I believe that the relative comparison between the two efforts is more accurate than the absolute number of 5,000 attached to RSA-129. The effort to break RSA-130 involved several thou-

sand machines working over six to eight months. A modern PC based on a Pentium II processor has been rated at between 88 and 220 MIPS on various benchmarks. The value of 88 was achieved on the Nsieve benchmark, an implementation of the Sieve of Eratosthenes. This benchmark is very close to what is performed by the Number Field Sieve. It would therefore seem that five to six such machines could break RSA-130 in one year. Yet this is incompatible with the claim that several thousand machines were required over six to eight months, even giving generous allowances for speed and memory differences. This gives reason to doubt the claim of 5,000 MIPS years for RSA-129.

tion instruction will run four times faster, and comparative benchmarks would conclude that it is a 40-MIPS machine. Even when a machine has such instructions, a particular compiler might not make them available to higher level languages; hence, machine performance becomes compiler-dependent.

TIME AND SPEED ARE NOT EVERYTHING

Traditionally, people have compared the difficulty of solving problems (not just cryptographic ones) by comparing the *time* required to solve them. Almost everyone ignores the fact that very frequently other resources can constrain the ability to solve a problem.

I wish to emphasize here that time is not the only resource required for key-breaking algorithms. In particular, the GNFS also requires large amounts of space. Indeed, in the current state of the art, space is a much more binding constraint than is time.

Table 2 shows the amount of memory required by NFS to break different sized keys. NFS is run in two stages. In the first stage, multiple machines independently perform sieving. The amount of memory given is the amount per machine. In the second stage, a very large sparse matrix must be solved. The data given is the amount of memory needed to store the matrix. Furthermore, solving the matrix efficiently requires a large Cray-class computer with fast static RAM, as opposed to the typical dynamic RAM found in most PCs and workstations.

As the table shows, breaking a 512-bit RSA key is barely feasible today. You might do it using about 3,000 SGI Indigo R10000 computers, each with 160 Mbytes of RAM, in about three months of sieving time followed by one to two weeks on a Cray-class machine with 10 Gbytes of RAM. This estimate should be accurate to within a factor of two.

In fact, an effort to break RSA-155, a 512-bit RSA challenge number (see the sidebar "Debunking MIPS Claims" for more on RSA challenge numbers), is under way. Breaking a 1,024-bit key is about six

million times harder in time complexity and about 3,000 times harder in space complexity. It is highly unlikely that we will see enough machines having 256 Gbytes of memory over the next 20 years to even begin the sieving phase of an attack on a 1,024-bit key. In addition, the memory needed to store the matrix is far beyond what can be achieved, let alone what anyone can afford. Cost is a major consideration here because static RAM is expensive.

TOWARD A BETTER METRIC

Given that the MIPS year as it is applied now is an inadequate way to measure key-breaking efforts, is a more effective measure feasible? I believe so. My approach still uses MIPS years, but it relates instructions to arithmetic operations.

1. Implement the algorithm and run it for perhaps several hours on the target platform.
2. Count the number of instructions executed during this time and the number of arithmetic operations. You can use profiling tools to do this. Let K be the ratio of total instructions to arithmetic operations. K is never less than one.
3. Determine the MIPS year equivalent for breaking a key with this algorithm by computing the number of arithmetic operations required (given by the time complexity function described next). Equate MIPS years to $(K \text{ times the number of arithmetic operations}) / (3.1 \times 10^{13})$.

Table 2. RAM needed to execute the Number Field Sieve algorithm.

Key size (bits)	Sieve memory (per machine)	Matrix memory (storage)
332	24 Mbytes	128 Mbytes
428	64 Mbytes	2 Gbytes
512	160 Mbytes	10 Gbytes
1,024	~256 Gbytes	~100 Tbytes

Table 3. Key sizes (bits) for public-key algorithms and arithmetic operations required for each size.

RSA*	Elliptic curve	Symmetric	Arithmetic operations (32-bit)
428 (RSA-129)	110	51	5.5×10^{17}
512	119	56	1.7×10^{19}
768	144	69	1.1×10^{23}
1,024	163	79	1.3×10^{26}
2,048	222	109	1.5×10^{35}

*The complexity of breaking DSA and RSA is essentially the same, so I include a column only for RSA.

Under this complexity measure, RSA-129 would require 17,700 MIPS years, as opposed to the reported 5,000.

Improved key size comparisons

This section gives better key size comparisons than what has appeared in the literature, based on using the suggested improved metric.

Table 3 gives key size comparisons between RSA, DSA, Elliptic Curve Discrete Logs (ECDL), and symmetric key (such as DES) algorithms as considered by time complexity only. These results compare the complexity by looking at the number of *arithmetic operations* needed to effect an attack. The table is purely relative in the sense that it does not attempt to attach a MIPS year measurement to the effort involved. A basic assumption of this table is that all arithmetic operations are 32-bit operations.

The table gives equivalent key sizes in bits for RSA, elliptic curve, and an arbitrary symmetric key algorithm. The table is derived by computing the number of arithmetic operations needed to break an RSA key, based upon the time-complexity function for the Number Field Sieve, which is the best known attack at present. This function is

$$e^{((c+o(1)) (\log N)^{1/3} (\log \log N)^{2/3})}$$

where $c = (64/9)^{1/3}$ and N is the RSA modulus. We do not have measurements of the $o(1)$ term for the General Number Field Sieve, so I assume a value of zero here. For the Special Number Field Sieve, the value for $o(1)$ is substantially less than 0.01 for N near 1,024 bits. The Special Number Field Sieve only works on numbers of special form such as $2^k - 1$, whereas GNFS works on all numbers.

The entry in the elliptic curve column is then derived by determining the key size that can be broken in the same number of arithmetic operations. This column assumes that the attack is parallel Pollard Rho, which is also the best known attack at present. The complexity function for this is

$$(\sqrt{\pi}) 2^{(k-1)/2} A(k)$$

where k is the number of bits in the size of the field

over which the elliptic curve is defined and $A(k)$ is the number of operations needed to add two points on the curve. $A(k)$ is typically (for a 32-bit machine) a small constant times $(k/32)^2$. This constant is implementation-dependent and may depend on the precise algorithm used, whether any precomputation was used, the type of basis for the finite field, whether projective or affine coordinates were used, and so on. For BSafe 4.0, the constant is approximately 450 for odd degree fields. Changing it by a factor of two changes the corresponding key size by 1 bit.

The symmetric column assumes that a block can be encoded in 284 arithmetic operations. Doubling this number would reduce the key size by 1 bit. RC6, a proposed algorithm for the Advanced Encryption Standard (AES), uses 284 operations. The time complexity function for breaking a symmetric key is, on average

$$(2^{k-1}) \times E(k)$$

where k is the key size and $E(k)$ is the time to encrypt a block.

Relying solely on MIPS estimates of machine speed and MIPS year estimates for key-breaking efforts can give misleading results when comparing key sizes for different cryptographic algorithms. MIPS ratings usually derive from standard benchmark suites. Using these ratings to then estimate machine speed on other problems can give misleading results. Such reliance can also underestimate the security of some algorithms because it implicitly assumes that time is the only constraint, whereas this discussion also shows that some of the best algorithms also require massive amounts of space. ❖

Robert D. Silverman is a senior research scientist at RSA Laboratories. In addition to cryptography, his interests include the design and analysis of computer algorithms, computational number theory, and large-scale distributed and parallel computing. He received an ABD in operations research from the University of Chicago. He is a member of the American Mathematical Society. Contact him at bobs@rsa.com.