The new SHA-3 software shootout

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

[1] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to
[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, Netherlands
tanja@hyperelliptic.org

1 Introduction

This paper introduces a new graphing mechanism to allow easy comparison of software performance of the SHA-3 candidates. The new mechanism concisely captures a large amount of performance data without oversimplifying the data.

We have integrated this graphing mechanism into our eBASH (ECRYPT Benchmarking of All Submitted Hashes) project. New graphs are automatically posted at the top of http://bench.cr.yp.to/results-sha3.html whenever the eBASH performance results are updated. This paper includes snapshots of these graphs, but readers are advised to check the web page for the latest updates. See http://bench.cr.yp.to for more information regarding eBASH.

For each function there is also a similar graph online comparing implementations of that function, showing in a concise way which implementations are slow or non-functional. Implementors can follow links from http://bench.cr.yp.to/primitives-sha3.html to find these graphs. Of course, users concerned about performance will reject slower implementations in favor of faster implementations, so the main shootout graphs reflect only the fastest implementations.

2 Understanding the graphs

It is tempting to report the software performance of a SHA-3 candidate as a single number: the time to hash a message. One could then summarize the performance of SHA-2 and the 5 SHA-3 candidates as 6 points in a one-dimensional graph.

However, as discussed in this section, the time to hash a message depends heavily on the message length and on the CPU used for hashing. A graph showing these dependencies is naturally three-dimensional: one axis shows message length, one axis shows the CPU, and one axis shows time. Furthermore, on the time axis there are actually 10 points rather than 6, as explained below.

The graphs in this paper put time on the horizontal axis and the platform on the vertical axis. Message length is on the page axis: the first, second, third, fourth, fifth, and sixth

This work was supported in part by the European Commission under Contract ICT-2007-216676 ECRYPT II and in part by NIST grant 60NANB10D263. Permanent ID of this document: e04ab717c1ba1be3e5e2fe4c6f8568a3. Date: 2012.02.19.

pages show long messages, 4096-byte messages, 1536-byte messages, 576-byte messages, 64-byte messages, and 8-byte messages respectively.

The graphs are longer horizontally than vertically, with a 4×3 format to fit typical projector screens. This document has a format that is higher than long, to fit typical printers, so we rotated the graphs 90 degrees to the right. In the following description we continue to say "horizontal" and "vertical" as if this rotation had not occurred: we expect readers to undo the rotation, turning the graphs 90 degrees to the left to look at them, so that the text inside the graphs is horizontal.

2.1 Vertical axis: architecture/microarchitecture/CPU/machine

An Apple iPhone 4 contains an Apple A4 CPU with a single ARM Cortex A8 core running at 1GHz (10^9 CPU cycles per second). A desktop computer bought at the same time might contain an AMD Phenom II X6 1100T CPU with six cores running at 3.3GHz ($3.3 \cdot 10^9$ CPU cycles per second on each core, for a total of $19.8 \cdot 10^9$ CPU cycles per second). It is not surprising that the second CPU can hash an order of magnitude more data per second than the first.

SUPERCOP measures cryptographic software on a single CPU core. Most hash functions, and in particular all SHA-3 candidates, fit within a single core, so hashing N separate messages on N cores scales in the obvious way. SUPERCOP also reports timings in CPU cycles, rather than microseconds or nanoseconds.

These steps drastically *reduce* the variability in measurements across CPUs. However, they do not *eliminate* the variability. Different CPUs have different "microarchitectures" carrying out different amounts of work in a cycle: CPUs vary in the number of fast arithmetic operations per cycle (typically between 1 and 3), the maximum size of those arithmetic operations (typically 32 bits or 64 bits), the availability of vector operations, the availability of AES arithmetic, etc. The impact varies from one hash function to another.

Our new graphs sort benchmarked machines according to their CPU microarchitecture. Each microarchitecture is given the same amount of vertical space; that space is partitioned evenly between the machines with that microarchitecture (but leaving slightly more space at the top and bottom of each microarchitecture). The obvious presumption is that machines with the same CPU microarchitecture will have the same hashing performance, so the points within a microarchitecture will be aligned vertically; this indeed happens in most cases, and the occasional deviations are immediately visible. The most common reasons for deviations are the following:

- The performance of C implementations is often compromised by compiler stupidity. The level of stupidity varies with the compiler version, which in turn varies from one machine to another. This effect is generally reduced for implementations that include assembly language or other microarchitecture-specific optimizations; an extreme defense against compiler stupidity is to submit a complete assembly-language implementation produced by a better compiler. All of the SHA-3 finalist teams have submitted microarchitecture-specific implementations, although not always for all architectures of interest, and not always for round 3; see in particular the discussion of JH below.
- Sometimes a hash function has to wait because CPU resources are being consumed by another process. Occasional interruptions are automatically filtered out (they do not affect

the quartiles), but "hyperthreading" interruptions produce a pervasive, hard-to-analyze slowdown. Machines known to have hyperthreading are marked with "threads" in red in our tables, and we now exclude them from our shootout graphs.
- Sometimes a hash function runs more slowly because a machine does not have the latest implementations. The graphs indicate the SUPERCOP version used for each machine, and exclude versions older than July 2011.
- Sometimes a hash function runs more quickly because a CPU is dynamically overclocked (while the cycle counter is not). Machines known to have dynamic overclocking are marked with "boost" in red in our tables, and we now exclude them from our shootout graphs.

Machines are labelled on the right side of each graph, and microarchitectures are labelled in a larger font on the left side of each graph. We organized microarchitectures by hand, with the goals of bringing similar microarchitectures together and reducing "jumps" in the graph. Machines within one microarchitecture are automatically sorted according to a particular metric of overall performance: for example, `trident` has an old operating system with an obsolete compiler, and automatically ends up at the bottom of the C2 65nm list.

JH deserves special note because the JH designer submitted reasonable round-2 JH implementations but no round-3 JH implementations. A third party submitted a slow round-3 JH implementation that has no microarchitecture-specific optimizations, and for some time this implementation was the only one reflected in the shootout graphs. We now have (from the first author of this paper) a reasonable round-3 JH implementation for amd64, and an independent team submitted a reasonable round-3 JH implementation for ARM, but these implementations have not been benchmarked on all machines yet. This is why the graphs show JH slowing down by more than a factor of 2 on many machines, compared to the best results on machines with the same microarchitecture. These slowdowns will disappear as more machines update their benchmarks.

2.2 Page axis: message length

It is not surprising that long messages take more time to process than short messages. These graphs report cycles per message byte, rather than cycles. This *reduces* the variability among different message lengths but does not *eliminate* it: all hash functions have overheads that become severe for short messages. The impact of these overheads varies from one hash function to another.

The selection of message lengths for these graphs matches the selection used for the ECRYPT Stream Cipher Project (eSTREAM), but we are investigating ways for readers to view similar graphs for many more message lengths. Consider, for example, two hash functions that each use 20 cycles per byte for 64-byte messages, and 80 cycles per byte for 8-byte messages. One cannot safely interpolate between these figures to predict the performance of an application hashing 40-byte messages with either hash function. Perhaps the first function handles messages in 32-byte blocks, and thus uses 32 cycles per byte for 40-byte messages. Perhaps the second function handles messages in 8-byte blocks but adds 6 blocks of finalization to each message, and thus uses only about 25 cycles per byte for 40-byte messages. The most common block size among SHA-3 finalists is 64 bytes, but there are several exceptions: `blake512` and `groestl512` (like `sha512`) use 128 bytes; `keccakc512` uses 136 bytes; `keccakc1024` uses 72 bytes.

Our first graph extrapolates long-message performance. The extrapolation is $1/B$ times the difference between the cycles to hash 4096 bytes and the cycles to hash $4096 - B$ bytes. Here B is 1904 for `keccakc512`, 2016 for `keccakc1024`, and 2048 for all other SHA-3-256/SHA-3-512 candidates, ensuring in each case that B bytes are an integer number of blocks.

2.3 Horizontal axis: time

The time axis is cycles per byte, as explained above. The fastest SHA-3 candidates are on the left: fewest cycles per byte. The slowest SHA-3 candidates are on the right: most cycles per byte. The scale is logarithmic, so a constant horizontal distance means a constant ratio in performance; powers of 2 are marked at equal intervals on the bottom of each graph.

Color, either on a display or on a printout, is important for the readability of these graphs. We use red $(1, 0, 0)$ for BLAKE, green $(0, 0.8, 0)$ for Grøstl, cyan $(0, 0.6, 0.6)$ for JH, purple $(0.8, 0, 0.8)$ for Keccak, dark blue $(0, 0, 1)$ for Skein, and black $(0, 0, 0)$ for SHA-2.

Normally there are 10 points marked on each horizontal line. These points consist of 6 crosses (connected by single lines) for SHA-256 and the five SHA-3-256 candidates, and 6 dots (connected by double lines) for SHA-512 and the five SHA-3-512 candidates. The JH proposals for SHA-3-256 and SHA-3-512, namely `round3jh256` and `round3jh512`, have identical performance. The Skein proposals for SHA-3-256 and SHA-3-512, namely `skein512256` and `skein512512`, also have identical performance to each other. This is why there are normally only 10 points visible rather than 12. The current ARM JH implementation includes only `round3jh256`, not `round3jh512`, producing two JH points for ARM microarchitectures, but the rightmost point will disappear once this is fixed.

Other candidates vary in performance between 256 bits and 512 bits. The Keccak proposals for SHA-3-256 and SHA-3-512, namely `keccakc512` and `keccakc1024`, are similar to Skein and JH in having essentially the same processing for each block, but `keccakc1024` handles fewer message bytes per block, making it approximately 1.9 times slower than `keccakc512`. The Grøstl proposals, namely `groestl256` and `groestl512`, have different state sizes, making the performance harder to predict; `groestl512` is consistently slower than `groestl256` but the ratio depends on the CPU. The BLAKE proposals, namely `blake256` and `blake512`, have different state sizes and different internal word sizes (similar to SHA-256 and SHA-512); `blake512` is consistently slower than `blake256` on 32-bit CPUs, but `blake256` is slower than `blake512` on most 64-bit CPUs.

Some of the points in the graphs are surrounded by horizontal lines. Each measurement is repeated many times; the line stretches from the first quartile of these measurements to the third quartile, and the point is plotted at the median. The lines are actually plotted around each point, but are usually not visible. Long horizontal lines should thus be taken as a warning: they indicate high variations in measurements. Don't trust any benchmarking system that fails to report variations in its measurements!

2.4 Alternate functions

All of the submissions have also specified 224-bit and 384-bit functions. There is no evidence of any interest in the performance of these functions. In all cases except for Keccak the

224-bit function has the same speed as the 256-bit function, and the 384-bit function has the same speed as the 512-bit function.

The Keccak documentation defines, and recommends, two more functions with the 256-bit and 512-bit output sizes, namely `keccak` truncated to 256 or 512 bits. These functions have the same performance as each other, about 6% slower than the `keccakc512` proposal for SHA-3-256.

The Skein documentation defines another function with 256-bit output size, namely `skein256256`. Early in the SHA-3 competition the Skein team pressured ASIC implementors, FPGA implementors, and microcontroller implementors to report area measurements for `skein256256` and omit area measurements for `skein512256`, since `skein512256` consumes about twice as much space. However, the Skein submission to NIST clearly proposes `skein512256` for SHA-3-256 and `skein512512` for SHA-3-512, and includes corresponding test vectors.

The graphs in this document focus solely on the proposals for SHA-3-256 and SHA-3-512, and omit alternate functions defined in the same documents.

Recent work has shown that at least two SHA-3 candidates benefit from tree modes on a single CPU core: specifically, `bblake256` and `keccakc512treed2` achieve better long-message speeds than `blake256` and `keccakc512`. These modes are not included in the current graphs.

3 A brief survey of architectures and microarchitectures

Four different architectures are included in the graphs: armeabi, ppc32, x86, and amd64. Software written in assembly language (by hand or by a compiler) is written for only one of these architectures and will not work on the others. (Note regarding names: AMD introduced the amd64 architecture; Intel copied the architecture and now refers to it as "Intel 64"; some operating systems refer to amd64 as "x86-64".)

Many more microarchitectures are included in the graphs. In some cases there are large performance differences between microarchitectures sharing the same architecture. Sometimes one microarchitecture includes useful additional instructions that will not work on other microarchitectures: for example, some Westmere and Sandy Bridge CPUs (and the very new Ivy Bridge and Bulldozer, not yet benchmarked) support AES instructions that are useful for Grøstl, while other amd64 microarchitectures do not. Furthermore, even when the same instructions work on all amd64 microarchitectures, they usually work at different speeds on different microarchitectures. Software can be written so that it will work on, e.g., all of the amd64 microarchitectures, but the best results are usually obtained by separate software optimized for each microarchitecture.

The best-known line of microarchitectures is Intel's series of high-power microarchitectures:

- amd64 C2 65nm. CPU examples: 2006 Intel Core 2 Duo E6300; 2007 Intel Core 2 Duo E4600.
- amd64 C2 45nm. CPU examples: 2007 Intel Xeon E5420; 2008 Intel Core 2 Duo E8400. This was not merely a "die shrink" (using improved 45nm lithography instead of 65nm); it also added new instructions, such as "PSHUFB" to shuffle the bytes of a 128-bit vector.
- amd64 Nehalem. CPU examples: 2008 Intel Core i7 920; 2010 Intel Xeon X7560.

- amd64 Westmere. CPU example: 2011 Intel Core i5-480M.
- amd64 Westmere+AES. CPU examples: 2010 Intel Core i5 M 520; 2010 Intel Xeon X5680. This microarchitecture introduced AES instructions.
- amd64 Sandy Bridge. CPU example: 2011 Intel Core i3-2310M. Sandy Bridge introduced 3-operand vector instructions, allowing (e.g.) "`a = b + c`" to be expressed as a single addition instruction rather than a copy followed by an addition.
- amd64 Sandy Bridge+AES. CPU example: 2011 Intel Core i5-2500K.

It is not clear whether Intel's future high-power microarchitectures will show similar fragmentation, some supporting AES instructions and some not. We have not found any relevant statements from Intel.

AMD has a competing series of high-power 64-bit microarchitectures:

- amd64 K8. CPU examples: 2005 AMD Opteron 875; 2006 AMD Athlon 64 X2.
- amd64 K10 65nm. CPU examples: 2008 AMD Opteron 8354; 2008 AMD Phenom 9550.
- amd64 K10 45nm. CPU examples: 2008 AMD Opteron 2376; 2010 AMD Phenom II X6 1100T.
- amd64 K10 32nm. CPU example: 2011 AMD A8-3850.

Both Intel and AMD also have low-power microarchitectures:

- x86 Atom. CPU examples: 2008 Intel Atom Z520 (2 watts); 2009 Intel Atom N280 (2.5 watts); 2011 Intel Atom Z670 (3 watts); 2012 Intel Atom Z2460 (1 watt?).
- amd64 Atom. CPU examples: 2009 Intel Atom D510 (13 watts); 2010 Intel Atom N455 (6.5 watts); 2011 Intel Atom D2500 (10 watts).
- amd64 Bobcat. CPU examples: 2011 AMD E-450 (18 watts).

Note that Intel is continuing to sell new 32-bit (x86) Atom CPUs, presumably because the 64-bit (amd64) architecture is more difficult to fit into very low power. See Figure 3.1 for the power consumption of Intel's 32-bit and 64-bit Atom CPUs.

The remaining microarchitectures in the shootout graphs are low-power microarchitectures from other CPU manufacturers:

- armeabi ARM11. CPU example: 2006 TI OMAP 2420 used in a Nokia N280 tablet.
- armeabi Tegra 2. CPU example: 2010 NVIDIA Tegra 2 used in a Samsung Galaxy Tab 10.1.
- armeabi Cortex A. CPU example: 2009 Freescale i.MX515. The Cortex A is the most common microarchitecture in low-power tablets and smartphones; for example, the Apple A5 CPU used in the iPad 2 has two Cortex A9 cores, and the Apple A4 CPU used in the original iPad has one Cortex A8 core, in both cases with a Cortex A microarchitecture.
- x86 Eden. CPU example: 2006 Via Eden ULV.
- ppc32 G4. CPU examples: 2001 Motorola PowerPC G4 7410 used in the Apple PowerMac G4; 2005 Motorola PowerPC G4 7447a. The current Freescale e600 line of embedded CPUs reportedly uses the same microarchitecture.

Anecdotal evidence suggests that 32-bit PowerPC and 32-bit MIPS are more common than ARM in many embedded applications, such as car CPUs and router CPUs. Many other embedded applications use lower-cost 16-bit and 8-bit CPUs.

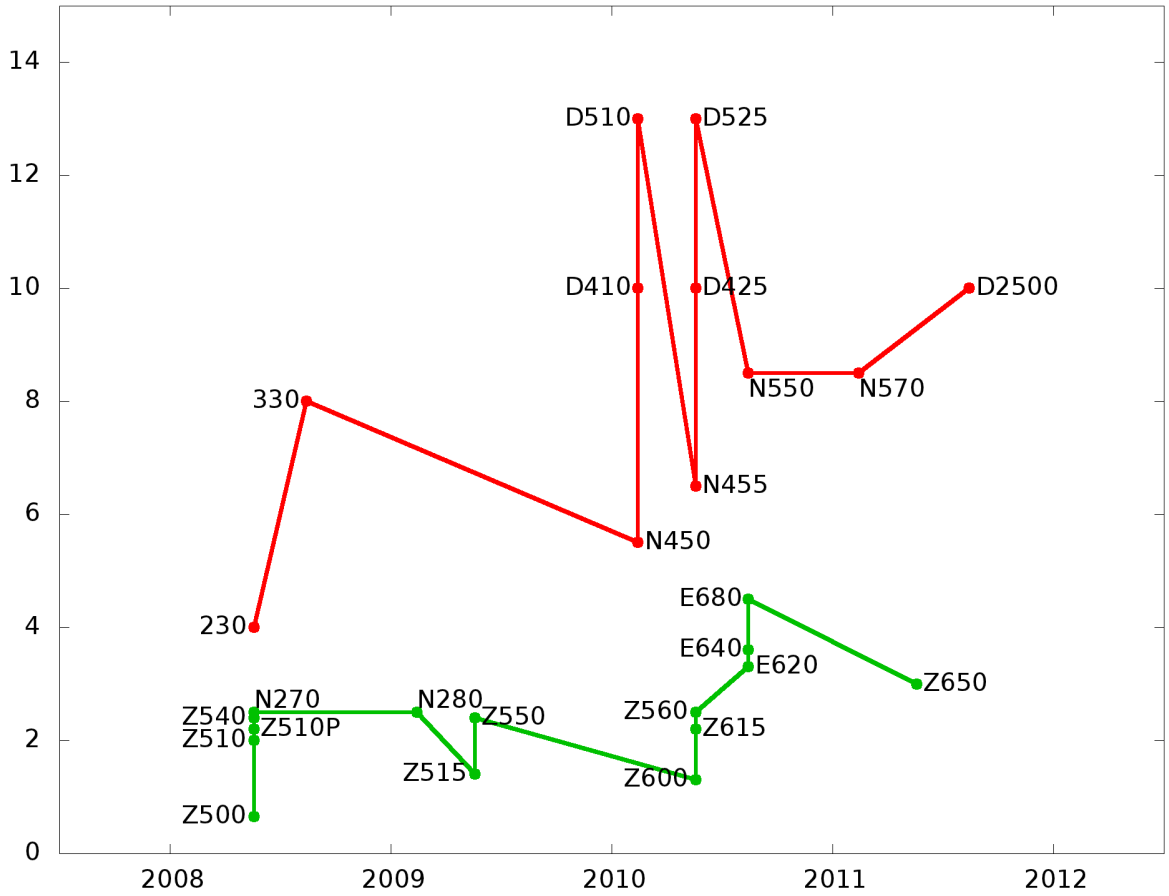
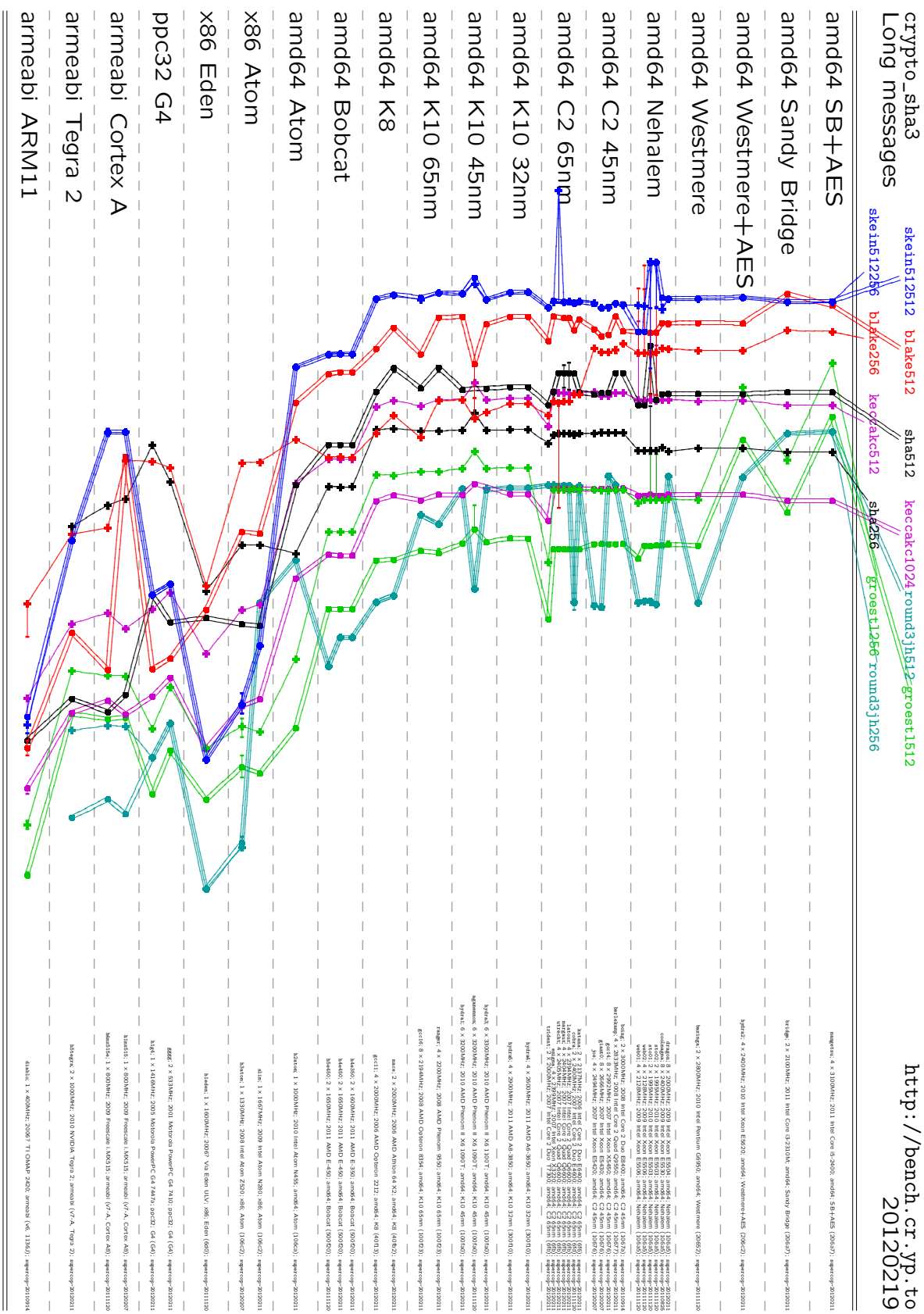


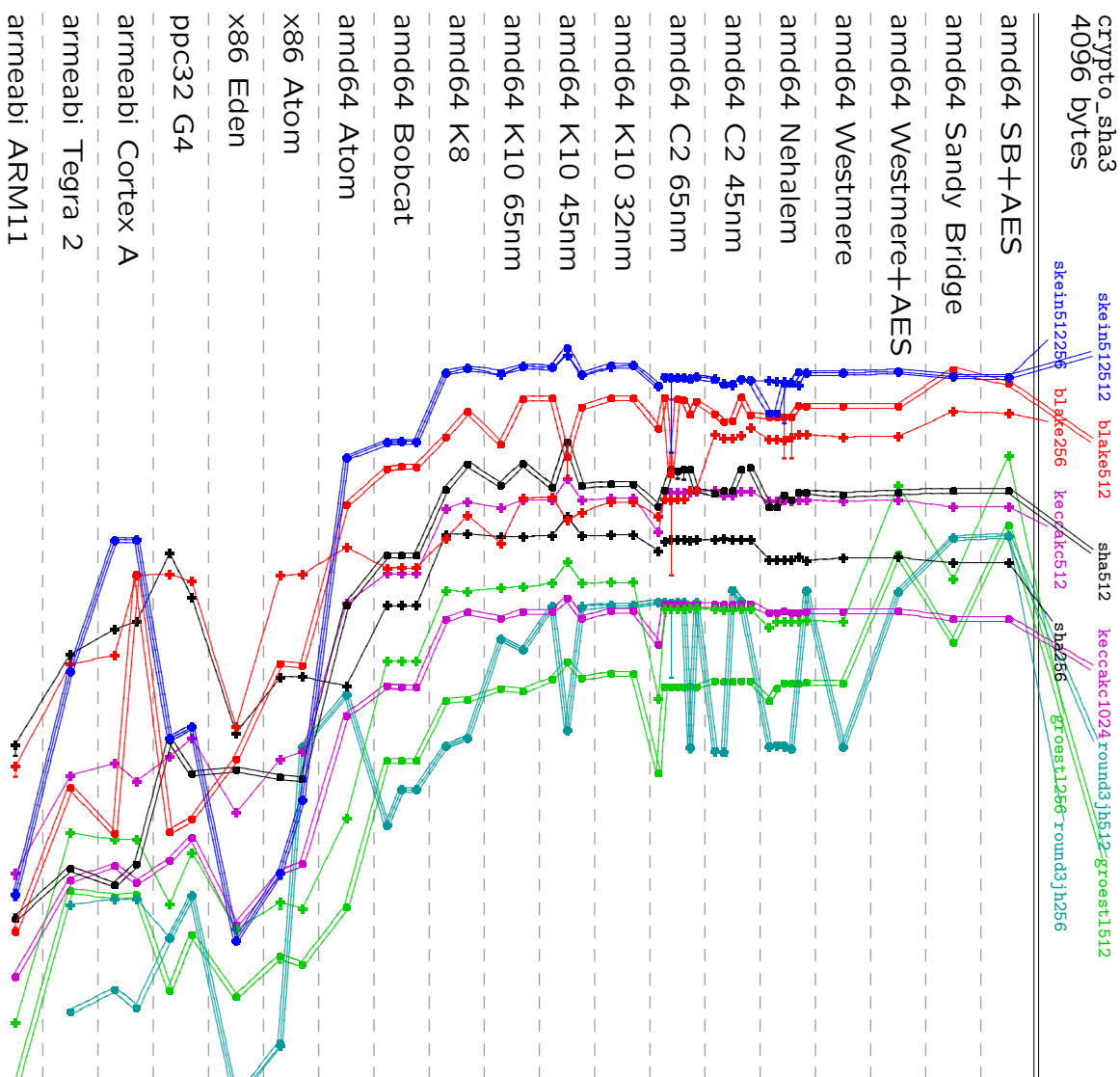
Fig. 3.1. Vertical axis: Thermal design power (TDP) of Intel Atom processors, in watts. Horizontal axis: Release date. Green curve (lower) shows 32-bit Atom processors. Red curve (higher) shows 64-bit Atom processors.

There are several other CPU architectures of interest. Examples: Fujitsu's K Computer, one of the largest supercomputers in the world, uses sparc64 CPUs. The PlayStation 3, the Xbox 360, and many supercomputers use ppc64 CPUs. ARM has announced future 64-bit ARM CPUs aimed at servers, although we expect that lower-power 32-bit ARMs will remain dominant in smartphones.

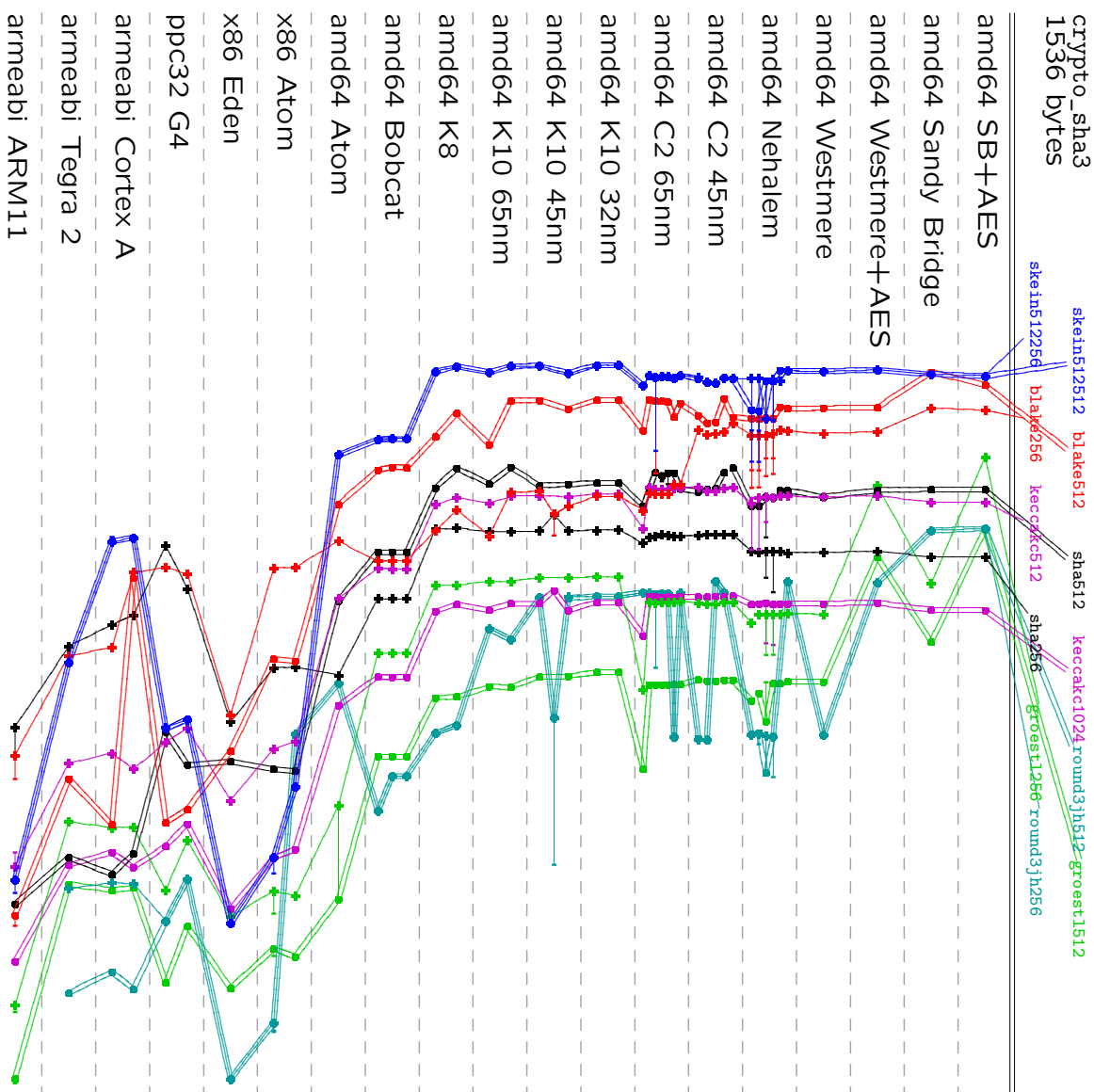
4 Comparison of SHA-3 finalists for long messages



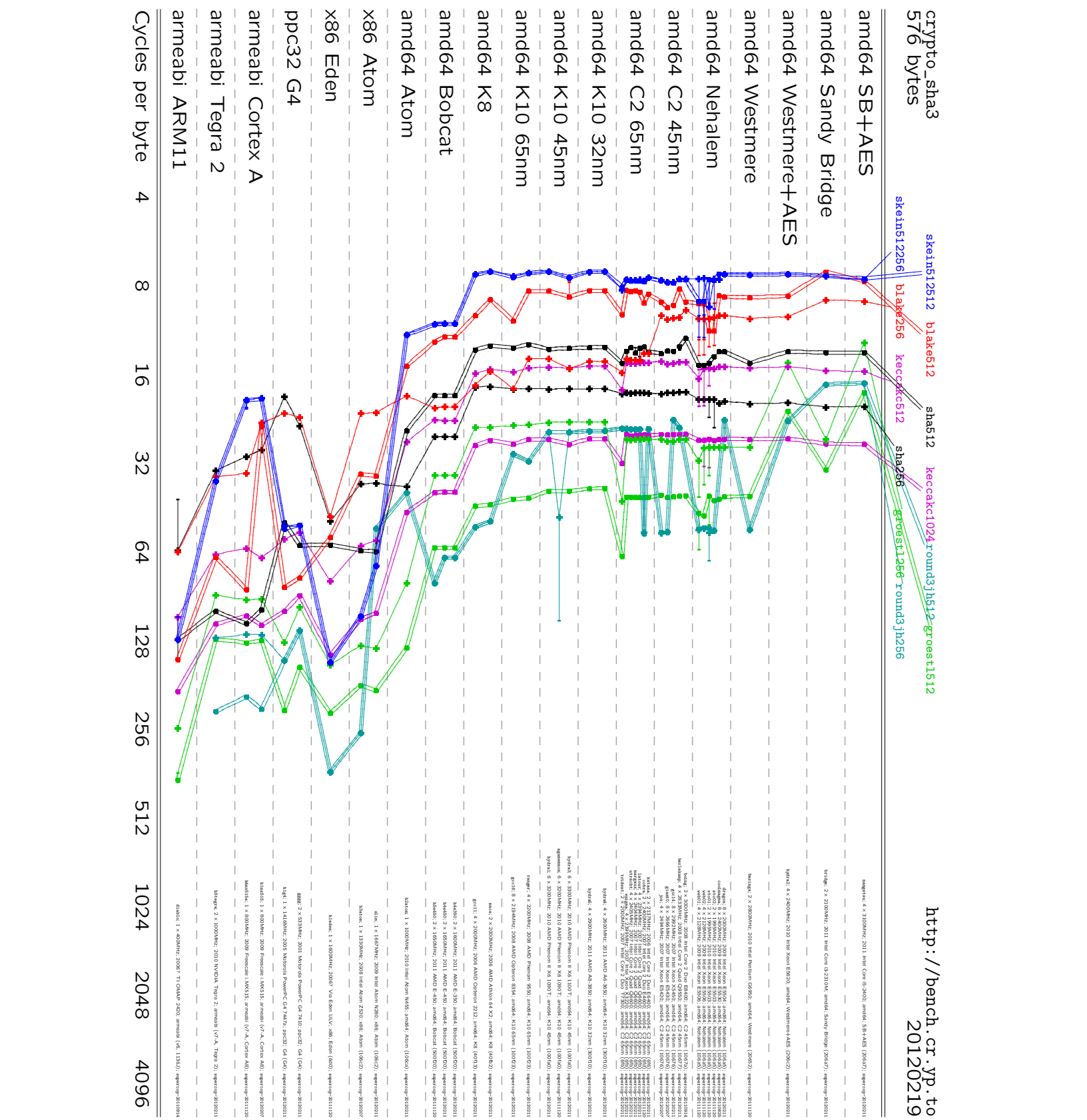
5 Comparison of SHA-3 finalists for 4096 bytes



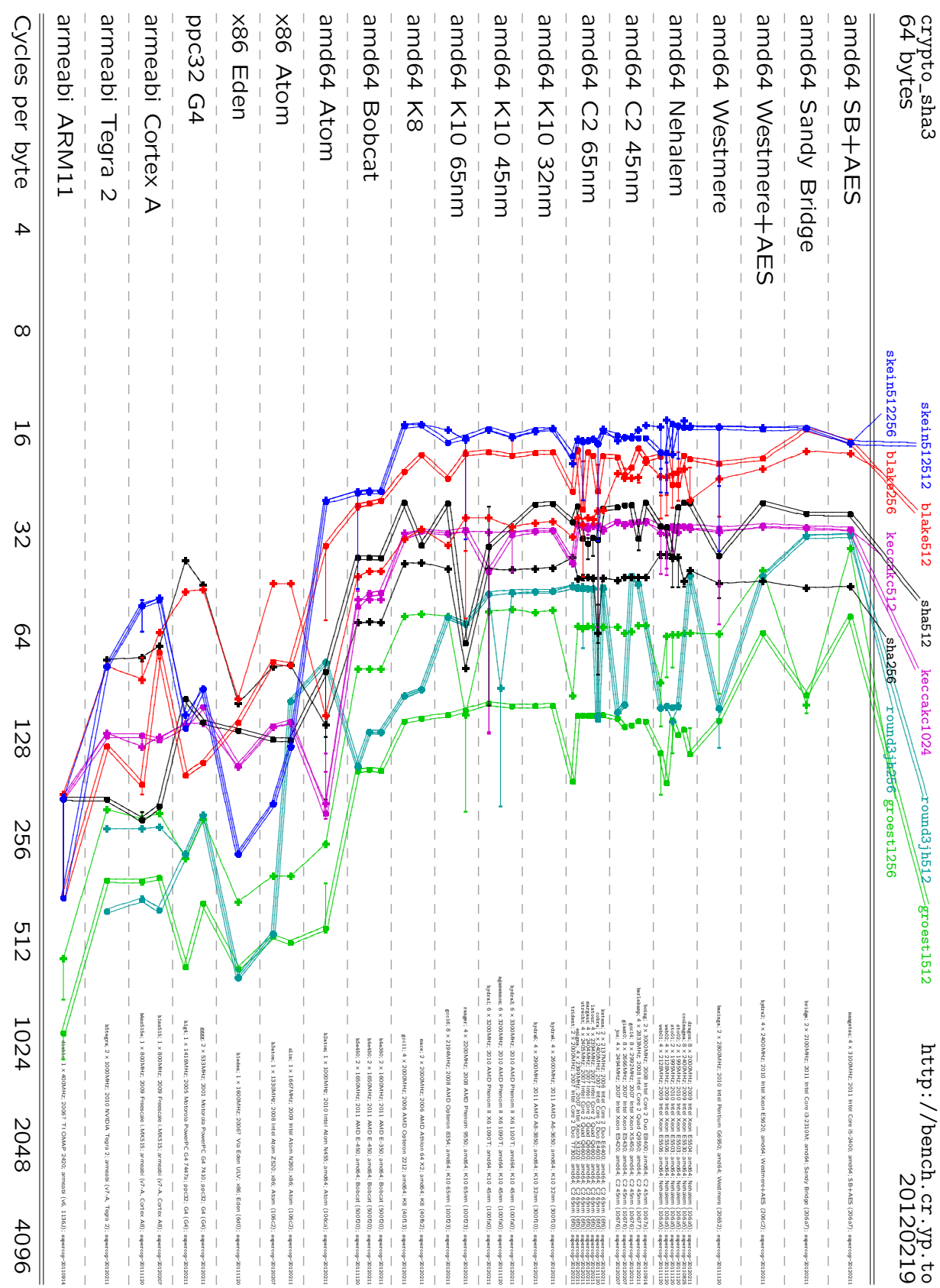
6 Comparison of SHA-3 finalists for 1536 bytes



7 Comparison of SHA-3 finalists for 576 bytes



8 Comparison of SHA-3 finalists for 64 bytes



9 Comparison of SHA-3 finalists for 8 bytes