1 Performance Comparison of SHA-3 Finalists

This section discusses how the finalist candidates perform when implemented in software for different computers, and in hardware circuits.

All of the SHA-3 finalist candidates, as well as SHA-2, have four variants with 224, 256, 384 and 512-bit message digest outputs. Skein and JH generate all four message digest sizes with the same compression function, and therefore, all four run at about the same rate. Keccak also uses a single compression function to generate all four output sizes, but the bigger the output, the smaller the message block that is processed by each compression function call, and therefore, the bigger the message digest, the slower Keccak runs. Blake, Grøstl and SHA-2 use two different compression functions, one for 512 and 384-bit message digests, and one for 256 and 224-bit message digests. The 512/384 bit ~~functions~~ and the 256/224 bit hash functions ~~digests~~ ~~vuariants~~ usually run at different speeds on the same platform.

Therefore, in the performance discussions, we will usually refer to:

- ~~Skein~~
- Blake-256 and Blake-512
- Grøstl-256 and Grøstl-512
- JH
- Keccak-224, Keccak-256, Keccak-384 and Keccak-512
- Skein
- SHA-256 and SHA-512

In the performance discussion, whenever the name of an algorithm is used without a specific digest size attached, then the statement applies for all four digest sizes.

1.1 Software Performance ~~Comparison~~

We will first describe the current computer systems that we expect to run SHA-3, and make some near term projections. Then we will discuss the performance differences found on these kinds of systems.

1.1.1 Computer Systems – the Current Playing Field

~~A majority of the cryptographic hash operations done today are probably implemented as computer programs written for some general purpose computer in either the computer's assembly language, or in some higher level language, or some combination of the two; and run on a general purpose computer. In this section we discuss the various types of computers that are commonly used today to execute hash functions, and the near term changes that we expect to see in them.~~

Naformátováno: Nadpis 1

Naformátováno: Odrážky a číslování

Naformátováno: Zarovnat do bloku

Komentář [SC1]: May want to insert a dash after the number later on.

Naformátováno: Zarovnat do bloku

Komentář [SC2]: Shall we just say "funtions" instead? Just to be consistent.

Naformátováno: Odrážky a číslování

Naformátováno: Zarovnat do bloku

Naformátováno: Nadpis 2

Naformátováno: Odrážky a číslování

Naformátováno: Zarovnat do bloku

Komentář [Meltem3]: Do we need this section? In the security section, we tried to be as brief as possible and did not include anything other than the results of the papers. This section is too detailed comparing to the security section.

Naformátováno: Nadpis 3

Naformátováno: Odrážky a číslování

~~This discussion will divide computers into *general-purpose computers* and *embedded computers*. General purpose computers are the computers that we recognize explicitly as computers, and they generally have a separate computer chip with various support chips for memory, displays and the like. We generally expect to be able to run compilers on these systems to develop applications for them. These include the common desktop, laptop, netbook and server computers that are ubiquitous today.~~

~~Embedded computers characteristically put RAM and ROM memory, a CPU and peripheral controllers and logic, often with other application specific logic on a single chip, and are incorporated into some specialized device, such as a router, a modem, a cell phone, music player, engine controller, or sensor. We usually develop software for embedded computers on a general-purpose host computer. At the high end, embedded computers are not very different from the general-purpose computers when it comes to hash algorithms, but at the low end, embedded computers are much less powerful, and have limited memory.~~

Komentář [SC4]: Don't we need an article for these items?

~~Flynn's well known taxonomy [Flynn72] of computer architectures defines four general computer architectures. Three of them broadly cover the cases of interest here:~~

- ~~SISD (Single Instruction stream, Single Data): one instruction stream, each instruction operating on one data item at a time. This may be called a scalar computer, and other machines that attempt to run one instruction stream, but attempt to launch more than one instruction at a time from that stream whenever possible, are called "super scalar."~~
- ~~SIMD (Single Instruction stream, Multiple Data): one instruction stream operating on multiple data objects. Instructions for an SIMD processor are often called vector instructions, and the processor is often called a "vector unit."~~
- ~~MIMD (Multiple Instruction stream, Multiple Data stream): a multiprocessor of loosely coupled computers, each in its own right has an SISD or MIMD machine (or both).~~

Komentář [SC5]: Or "is"?

~~From the introduction of general-purpose digital computers in the 1950s until the 1990s, computers were primarily SISD machines, except for a relatively few very expensive "mainframe" and "super computers" that included some level of SIMD or MIMD structure. But by the late 1990s (about the time of the AES competition and the development of SHA-2), SIMD "vector units" were added to the desktop computers and servers of the era. Such vector units are now integral to most laptop, desktop and server computers, and are also used in some tablet computers, and smartphones. Vector units are commonly used for signal processing, including video and audio CODECs, as well as for graphics, image processing and cryptography. These vector units are one of the major areas where computer architectures are rapidly evolving.~~

~~In addition, some desktop and laptop computers, as well as dedicated gaming machines use separate SIMD graphics coprocessors, which can be very powerful instruments for cryptographic computation. However, the main cryptographic use for these units are as platforms for the massive computations required by cryptographic attacks, which often~~

~~are highly parallelizable. Since these graphics coprocessors are not normally used to protect data, we did not consider the use of these units in this study.~~

~~Computers are also classified by their instruction sets as a Complex Instruction Set Computer (CISC), or a Reduced Instruction Set Computer (RISC). CISCs have often been implemented by "microprogramming" the instruction set that compilers and assemblers output and the computer executes. By the late 1990s the dominant instruction set for desktop computers was the "x86" architecture, an instruction set implemented by several vendors. The x86, a CISC, used 32-bit arithmetic and logic instructions, had a 32-bit virtual address space, and had 32-bit registers; it had grown in stages from a family of early 8-bit microcomputers. Because of its importance and ubiquity, performance on the x86 architecture played a major role in the AES competition and selection process.~~

~~The late 1990s also saw the introduction of the 64-bit variants of the x86 instruction-set architecture, and the variant that has prevailed is called the AMD64 (or IA64) instruction-set. Now most of the CPUs used by laptop, desktop and server computers implement both the x86 in a 32-bit mode and the AMD64 in a 64-bit mode. At first, relatively little 64-bit software was available, but now the current versions of major operating systems support 64-bit mode, and operation in the 64-bit mode is rapidly displacing the 32-bit mode. The movement to 64-bit operations was anticipated in SHA-2: SHA-256 used 32-bit operations, while SHA-512 uses 64-bit operations.~~

Komentář [SC6]: Check for the consistent use of this dash later.

~~Current AMD64 instruction-set computer chips are often extremely complex, with one to eight separate "cores," each effectively an independent computer able to run its own program stream; and each with several execution units (a general purpose arithmetic logic unit or ALU, a floating point unit and also usually a vector unit), each usually capable of simultaneously executing instructions, and each "pipelined" so that instructions are executed in stages, and a new instruction can potentially be launched at each processor cycle. ALUs in fact are often capable of simultaneously launching two instructions at each clock cycle, while various implementations my execute instructions out of order or speculatively. Much of this is done to automatically execute as much of the program as possible in parallel, even though the program is written as a series of serial instructions.~~

Komentář [SC7]: This is a pretty long sentence. For readability, it may help to break this up a bit.

~~The vector units are now also integral to most x86/AMD64 cores and allow the explicit programming of parallel operations, where the same operation is performed on vectors of similar data types. Vector units are commonly used for signal processing, including video and audio CODECs and image processing. Since the introduction of vector instruction sets in x86 and AMD64 computers in the 1990s, they have gone from using a set of eight 64-bit registers (each unit capable of performing eight one-byte operations, four 16-bit operations, or two 32-bit operations in parallel on a register) to as many as sixteen 256-bit registers (each unit capable of two 128-bit, four 64-bit, eight 32-bit, sixteen 16-bit or thirty-two byte operations in parallel).~~

~~In the late 1990s, during the AES competition, several RISC processors, most of them with a 32-bit word orientation, were available for everything from embedded controllers at the low end, to powerful servers at the high end. In contrast to a CISC machine, RISC~~

~~machines typically have a simpler instruction set, designed for implantation in a pipelined arithmetic logic unit (ALU) that, if all goes well, can launch a new scalar instruction at every clock cycle. RISC machines characteristically directly decode and execute instructions in logic, rather than by micro programming on some underlying machine.~~

~~Today, RISC computers have virtually disappeared in servers, desktop computers and laptop computers, although several legacy RISC architectures remain in current use as controllers in embedded applications, and in gaming consoles. However, the Advanced RISC Machine (ARM) architecture now dominates applications such as smart phones, tablet computers, music players, and many embedded computer applications. ARM machines today are 32-bit machines, with a 16 × 32-bit register file, a fixed 32-bit instruction width (supplemented for some applications by 16-bit "thumb" instructions) usually implemented with a pipelined ALU that attempts to start one instruction per clock. Various ARM "cores" are licensed to many vendors and incorporated in a wide variety of ASICs, with a vast range of performance levels. Moreover, the ARM is now seen as a potential rival of the x86/AMD64 processors in areas where the X86/AMD64 architecture currently dominates, including server farms, and laptop or netbook computers, while low end x86/AMD64 processors are attempting to gain a toehold in the upper end of the main ARM application space, particularly for tablet computers and smartphones.~~

~~The argument for ARM based processors in servers is that the simpler ARM RISC architecture is thought to require less electrical power than an equivalent more complex AMD64 architecture, and electrical power consumption (and the resulting heat) seems the most fundamental limit on the size and computing power of big server farms. Still, at this point, the use of ARM processors in server farms seems more theoretical than actual. The recently announced ARMv8 [Grisen 11] is a 64-bit ARM extension with 64-bit registers that will allow the 32-bit ARM applications to be executed in a 64-bit OS. It will include instruction level support for AES, SHA-2 and a vector unit and may become a direct challenger to the X86/AMD64 architecture even at the high end of the scale; but so far the parts are not commercially available, and we have no performance data on any of the SHA-3 finalists when implemented on ARMv8 processors.~~

~~Finally there are very small computers embedded in process controllers, sensor, servo systems, smart cards, some RFID tags and many other applications. These "embedded microcontrollers" may have 8 or 16 bit word processors, or be "stripped down" versions of the ARM 32-bit processor. In many cases these computers won't do any hashing at all, however, in some cases, for example smart cards, hashing may be an important part of what they do. They typically have relatively little computational power, and often are constrained by a battery power supply or the power that can be coupled to the device electromagnetically, and, above all, they often have a very small RAM that all their functions must share.~~

We ~~can then~~ divide the space of commercially available computers that we have SHA-3 finalist performance data for into five categories~~into five classes of machines~~:

Naformátováno: Zarovnat do bloku

AMD64: These machines~~, described above,~~ are general purpose Complex Instruction-Set Computers (CISC) ~~general-purpose computers~~ with a 64-bit word orientation and Single Instruction Multiple Data (SIMD) vector units. They all run the 64-bit oriented AMD64 instruction set architecture (ISA). AMD64 machines predominate today in desktop, laptop and netbook computers as well as servers of all sorts. They typically have very large RAM memories, and increasingly have two or more independent "cores", each capable of independently executing a program thread. Each core typically has a general-purpose superscalar Arithmetic and Logic Unit (ALU) supporting 64-bit word operations and able to simultaneously launch two instructions at each clock cycle. They also have a vector unit in each core. Since vector units were introduced for the x86 architecture in the 1990s, they have gone in stages from eight 64-bit registers to thirty-two 256-bit registers for the latest announced products. The most recent machines have vector units that can simultaneously execute eight 32-bit operations or four 64-bit operations in a 256-bit vector register. Most desktop, laptop and server computers sold today are AMD64 machines.

Naformátováno: Zarovnat do bloku

X86: These are the 32-bit predecessors of the AMD64 computers. Many legacy systems run on X86 computers, which now usually are AMD64 machines operating in the X86 mode. Most fairly recent examples include a vector unit and a super-scalar ALU.

ARM- NEON: These machines run a relatively high-end implementation of the ARM ISA with a vector unit. The NEON vector instruction set uses registers that can be viewed as thirty-two 64-bit registers or sixteen 128-bit registers. Many of the SHA-3 finalists benefit significantly from 64-bit instructions or bit slice implementations on wider words, and run markedly faster on the NEON equipped ARM machines.

32-bit RISC: These Reduced Instruction-Set Computers (RISC) are scalar or super-scalar machines that are typically used today in a wide range of applications from smart phones, tablet computers, and appliances such as GPS units and music players, to controllers and sensors embedded in many products. By far ~~T~~the most widely used RISC ~~instruction set~~ ISA is the ARM ~~ISA~~, which is widely licensed in a variety of "cores" that are incorporated in application specific integrated circuits. Other legacy 32-bit RISC ISAs studied in the SHA-3 selection include the PowerPC (PPC), and MIPS ISAs.

~~***ARM- NEON:*** These machines run a relatively high-end implementation of the ARM ISA with a vector unit. The NEON vector instruction set uses registers that can be viewed as thirty-two 64-bit registers or sixteen 128-bit registers. Many of the SHA-3 finalists benefit significantly from 64-bit instructions or bit slice implementations on wider words, and run markedly faster on the NEON equipped ARM machines.~~

Embedded Microcontrollers: These are small computers, typically included on an Application Specific Integrated Circuit (ASIC) with memory and other application specific logic. In this category the primary constraint is usually RAM memory, although power may be another constraint. This ~~may be~~is the most diverse category, and there are a number of ~~computer architectures~~ISAs, including the 32-bit ARM, and sixteen and eight-bit microcontrollers. Applications for such computers include smart cards, sensors, smart meters, servo controllers, some RFID tags and a plethora of potentially networked appliances.

1.2 ~~SHA-3~~ Candidate Software Performance Studies

Naformátováno: Nadpis 2

Naformátováno: Odrážky a číslování

Several studies have been done that compare all SHA-2 and the SHA-3 finalists using programs written by a single programmer or a small team for a specific platform or language or with a specific design goal. These include studies on optimized Java code run on a current AMD64 computer [Hanser12] and in optimized assembler on the ARM11 processor [Yang12], which is widely used in smart phones and tablet computers, but does not include the NEON vector engine. However, the vast bulk of the available SHA-3 finalist performance data was provided by two cooperative projects, eBASH (ECRYPT Benchmarking of All Submitted Hashes) [Bernstein], and XBX [Wenzel-Benner 12]~~:~~

Komentář [Meltem8]: Is this list complete ?

~~eBASH (ECRYPT Benchmarking of All Submitted Hashes) [Bernstein], which focused on general purpose computers where it was possible to compile and run a program on the same computer. eBASH was supported by the Virtual Applications and Implementations Research Lab (VAMPIRE), a part of the ECRYPT II project. Tanja Lange, of the University of Eindhoven, and Daniel Bernstein, of the University of Illinois, Chicago, organized and led eBASH, which was the third such cryptographic benchmarking effort organized under VAMPIRE to measure the performance of cryptographic algorithms.~~

~~eBASH summary "shootout" results are presented for the fastest implementation found for each test machine, after much experimentation with setting compiler options and the like, to optimize the performance of each algorithm. Fastest means processes an input message byte in the smallest number of processor cycles. Memory requirements of the implementations are not studied. The eBASH SHA-3 "shootout" summaries include data for fifteen AMD64 processor models, two X86 processor implementations, the PPC G4, four different ARM core designs and one MIPS32 implementation. In many cases, the eBASH benchmarks were run on several different examples of the same general processor model.~~

Komentář [Meltem9]: I suggest combining these two paragraphs with section "eBash Performance measurements of general purpose computers".

- ~~XBX [Wenzel-Benner 12] focused on embedded computers where it is usual to compile or assemble code on some development system, and then run it on a small computer or "microcontroller." The effort was organized for the SHA-3 competition by Christian Wetzel-Benner and Jens Gräf. They collected and measured SHA-3 implementations on eight embedded platforms. The website for the effort is at: http://xbx.das-labor.org/trac/wiki. Some of the 32-bit ARM processors benchmarked on XBX were as powerful as some of the machines tested by eBASH, and included vector units, while others were 8- and 16-bit machines characteristically used with very little memory.~~

Naformátováno: Odrážky a číslování

Komentář [Meltem10]: Similar to my previous comment, I suggest combining this paragraph with the "Embedded Microcontrollers" section.

1.2.1 eBASH: ~~Performance Measurements of~~ General-Purpose Computers

Naformátováno: Nadpis 3

Naformátováno: Odrážky a číslování

The eBASH homepage is found at: http://bench.cr.yp.to/ebash.html. During the course of the SHA-3 competition, a large number of hash functions were benchmarked on general purpose computers, and much data on all of them can be found on the eBASH site. The best comparative presentation of the data for the SHA-3 finalists and SHA-2 is the "shootout" graphs found at: http://bench.cr.yp.to/results-sha3.html, and we made extensive use of this data, which includes six graphs that summarize algorithm performance for various message lengths. The eBASH SHA-3 "shootout" summaries include data for fifteen AMD64 processor models, two X86 processor implementations, the PPC G4, four different ARM core designs and one MIPS32 implementation. In many cases, the eBASH benchmarks were run on several different examples of the same general processor model.

Naformátováno: Zarovnat do bloku

Komentář [Meltem11]: I suggest combining these two paragraphs with section "eBash Performance measurements of general purpose computers".

The six summary graphs, as of May 22, 2012, are reprinted in Figures A1-A6. Only the 512-bit and 256-bit ~~algorithms~~ variants are plotted. On the summary graphs, only the best (fastest) implementation of all of the many implementations tested is reported, for each test system.

Throughput is stated in machine cycles per byte, and the fewer cycles the better the performance. The fastest performance of the best algorithms on the latest machines is on the order of about 6 cycles per byte.

Naformátováno: Zarovnat do bloku

One other feature of the eBASH website deserves special attention here. At http://bench.cr.yp.to/primitives-sha3.html, there is a table labeled "Which hash functions are measured? (SHA-2/SHA-3 excerpt)" is a table of all the variants of all the SHA-3 finalists, plus SHA-512 and SHA-256, and three "tree modes" of Blake and Keccak that implement i-thread parallel implementations of the hash algorithms. ~~By clicking on the name of the algorithm the reader may see a graph of the performance of all the implementations tested for that algorithm.~~ The implementations that depend heavily on vector units are often given names that identify the type of vector unit: sse, avx, mmx for AMD64 or x86 machines, or NEON for ARM machines.

We categorize the computers used in eBASH into four of the five groups described above:

- AMD64: use the AMD64 ISA and generally include a vector unit.
- X86: use the 32-bit X86 ISA and may include a vector unit.
- 32-bit RISC: use the following 32-bit RISC ISAs: ARM, MIPS or PPC. A vector unit is not used.
- ARM-NEON: use the 32-bit ARM ISA with the NEON vector unit.

Komentář [Meltem12]: I suggest to change the ordering of 32-bit RISC and ARM-NEON for consistency. Figures and the next section use a different ordering.

Naformátováno: Zarovnat do bloku

Komentář [Meltem13]: In Figure X and Y, colors are used. I think we should give a color coding. Or how about removing the colors?

To try to visually distil the complex graphs of the eBASH into a simpler presentation, we categorize performance of each algorithm to *high*, *medium* or *low* for the four different categories of computers in Figure x and y~~, in a manner somewhat analogous to grading papers into different letter grades~~. Figure x is for long messages (greater than 4096 bytes), while Figure y illustrates shorter messages of 64 bytes. These are based on the

Naformátováno: Písmo: Kurzíva

Naformátováno: Písmo: Kurzíva

Naformátováno: Písmo: Kurzíva

eBASH “shootout” graphs for six different sizes of messages (long, 4096-bytes, 1536-bytes, 576-bytes, 64-bytes and 8-bytes). These are essentially “eyeball” judgments of relatively noisy data; the reader can study Figures A-1 through A-6, the actual detailed eBASH shootout plots, to see how well our categorizations fit the data. The data for JH in particular seems erratic: this is apparently because the fastest JH vector code was completed very late, and had been run on some but not all the benchmark machines. This makes little comparative difference, because even the fastest JH implementations are among the slowest of the algorithms. For each category of machine, the performance range of the algorithms seems to be between two and three octaves, that is, the fastest algorithms seem to be about four to eight times the speed of the slowest.

1.2.1.1 ~~Performance on~~ Long Messages

Naformátováno: Nadpis 4

Naformátováno: Odrážky a číslování

Naformátováno: Zarovnat do bloku

Naformátováno: Písmo: Tučné

AMD64: Skein (all sizes) and Blake-512 are consistently the fastest algorithms and the only two algorithms that generally are faster than SHA-512 on AMD64 platforms. Blake-256 is also fast on newer AMD64 platforms with larger vector register files [Neves 12], but falls off on older machines. SHA-512, SHA-256 and Keccak-256 are in the medium range averaging about half the speed of the high group, along with Grøstl-256 and Grøstl-512 on very new machines with the AES-NI instructions. JH also makes the medium group for very recent AMD64 processors, but not older processors. Grøstl-256, Grøstl-512 (no AES-NI) and Keccak-512 and JH are in the low group, with performance only around ¼ of that of the high performance group.

X86: The high group here is Blake-256, Skein, and SHA-256, all ARX type algorithms. The low group is Keccak-512, and Grøstl. Although Skein does well, algorithms that use a 32-bit word may have an advantage on these 32-bit machines.

ARM - NEON*:* The high group is Blake, Skein and SHA-256, the medium is JH and Keccak-256, while the low group is Grøstl, Keccak-512 and SHA-512. The ability of the NEON vector unit to do 64-bit operations probably helps Skein and Blake-512 here.

32-bit RISC: The high performance algorithms are Blake-256 and SHA-256, with Grøstl, Keccak-512 and JH at the low end.

Algorithm	AMD64			X86			ARM-NEON			32-bit RISC		
	High	Med	Low	High	Med	Low	High	Med	Low	High	Med	Low
Blake-512	✓				✓		✓				✓	
Blake-256	✓	✓		✓			✓			✓		
Grøstl-512		✓*	✓			✓			✓			✓
Grøstl-256		✓	✓			✓			✓			✓
JH			✓		✓	✓		✓				✓
Keccak-512			✓			✓			✓			✓
Keccak-256		✓			✓	✓		✓			✓	

Skein	✓			✓	✓		✓				✓	
SHA-512		✓			✓				✓		✓	
SHA-256		✓		✓			✓			✓		

Table x - eBASH performance comparison for long (> 4096-byte) messages

* Only on processors with AES NI instructions

1.2.1.2 ~~Performance on~~ 64-byte Messages

This comparatively small message size starts to show the effects of both fixed per message overhead and of different input block sizes, which range from 512 to 1088 bits. Average cycles per byte seem to be about twice that of long messages, but the ordering of things has changed remarkably little. At this message size Keccak-512 looks about as good in comparison to others as it ever will, since 64 bytes just fits, with padding, in a single message block and it has no extra finalization round, while other algorithms either need two blocks or process a significantly larger 128-byte message block.

AMD64: As with long messages, Blake and Skein lead all others. Blake-256 is fast on new machines with large vector register sizes but drops down into the medium level for older AMD64 machines. Keccak-512 is now in the medium level with Keccak-256, because its 576-bit input block inputs the entire message in 1 block ~~and therefore runs as fast as Kecak-256~~. Grøstl-512 falls back to the low category even with the AES-NI instructions, probably because of the extra overhead of the final “blank” round. Grøstl-256 stays in the medium group with SHA-256 and SHA-512.

X86: Blake-256, Skein and SHA-256 are fast on this 32-bit word ISA. With no AES hardware support Grøstl-256 and 512 shows low performance along with Keccak-512 and SHA-512.

ARM-NEON: Blake-256, Blake-512 and SHA-256 are fast, while Grøstl-256, Grøstl-512 and SHA-512 are the slowest.

32-bit RISC: The two algorithms that use 32-bit modular addition extensively are fast, while Grøstl-256, Grøstl-512 and JH are slow.

Komentář [Meltem14]: I prefer a more compact table, so I combined the tables in Figure x. We may do the same for Figure y, if you prefer. I removed the colors, but we can put them back.

Naformátováno: Nadpis 4

Naformátováno: Odrážky a číslování

Komentář [SC15]: vector register files?

Naformátováno: Písmo: Kurzíva

Naformátováno: Doleva

Naformátováno: Písmo: Kurzíva

Naformátováno: Písmo: Kurzíva

Naformátováno: Doleva

AMD64

Algorithm	Relative Performance		
	High	Med	Low
Blake-512	🟦		
Blake-256	🟦	🟩	
Grostel-512		▨	🟥
Grostel-256		🟩	🟥
JH			🟥
Keccak-512			🟥
Keccak-256		🟩	
Skein	🟦		
SHA-512		🟩	
SHA-256		🟩	

X86

Algorithm	Relative Performance		
	High	Med	Low
Blake-512		🟩	
Blake-256	🟦		
Grostel-512			🟥
Grostel-256			🟥
JH		🟩	🟥
Keccak-512			🟥
Keccak-256		🟩	🟥
Skein	🟦	🟩	
SHA-512		🟩	
SHA-256	🟦		

▨

Komentář [Meltem16]: The colors encoding should be included to the caption.

ARM ~~with~~ NEON

Algorithm	Relative performance		
	High	Med	Low
Blake-512	🟦		
Blake-256	🟦		
Grostel-512			🟥
Grostel-256			🟥
JH		🟩	
Keccak-512			🟥
Keccak-256		🟩	
Skein~~-512~~	🟦		
SHA-512			🟥
SHA-256	🟦		

32-bit RISC ~~w/o vector unit~~

Algorithm	Relative performance		
	High	Med	Low
Blake-512		🟩	
Blake-256	🟦		
Grostel-512			🟥
Grostel-256			🟥
JH			🟥
Keccak-512			🟥
Keccak-256		🟩	
Skein		🟩	
SHA-512		🟩	
SHA-256	🟦		

Naformátována tabulka

Fig x - eBASH performance comparison for long (> 4096-byte) messages

AMD64

Algorithm	Relative Performance		
	High	Med	Low
Blake-512	🟦		
Blake-256	🟦	🟩	
Grostel-512			🟥
Grostel-256		🟩	🟥
JH		🟩	
Keccak-512		🟩	
Keccak-256		🟩	
Skein	🟦		
SHA-512		🟩	
SHA-256		🟩	

X86

Algorithm	Relative Performance		
	High	Med	Low
Blake-512		🟩	
Blake-256	🟦		
Grostel-512			🟥
Grostel-256			🟥
JH		🟩	
Keccak-512			🟥
Keccak-256		🟩	
Skein-512	🟦		
SHA-512			🟥
SHA-256	🟦		

ARM ~~with -~~ NEON

Algorithm	Relative performance		
	High	Med	Low
Blake-512		🟩	
Blake-256	🟦		
Grostel-512			🟥
Grostel-256			🟥
JH		🟩	🟥
Keccak-512		🟩	
Keccak-256		🟩	
Skein		🟩	
SHA-512		🟩	
SHA-256	🟦		

32-bit RISC ~~w/o vector unit~~

Algorithm	Relative performance		
	High	Med	Low
Blake-512		🟩	
Blake-256	🟦		
Grostel-512			🟥
Grostel-256			🟥
JH			🟥
Keccak-512		🟩	
Keccak-256		🟩	
Skein		🟩	
SHA-512		🟩	
SHA-256	🟦		

~~Fig. y – eBASH relative performance comparison for 64-byte messages~~

Naformátováno: Doleva

„ Grøstl-256 ~~Algorithm~~	~~AMD64~~			~~X86~~			~~ARM-NEON~~			~~32-bit RISC~~		
	~~High~~	~~Med~~	~~Low~~	~~High~~	~~Med~~	~~Low~~	~~High~~	~~Med~~	~~Low~~	~~High~~	~~Med~~	~~Low~~
~~Blake-512~~	~~✓~~				~~✓~~		~~✓~~				~~✓~~	
~~Blake-256~~	~~✓~~	~~✓~~		~~✓~~			~~✓~~			~~✓~~		
~~Grøstl-512~~		~~✓*~~	~~✓~~			~~✓~~			~~✓~~			~~✓~~
~~Grøstl-256~~		~~✓~~	~~✓~~			~~✓~~			~~✓~~			~~✓~~
~~JH~~			~~✓~~		~~✓~~	~~✓~~		~~✓~~				~~✓~~
~~Keccak-512~~			~~✓~~			~~✓~~			~~✓~~			~~✓~~
~~Keccak-256~~		~~✓~~			~~✓~~	~~✓~~		~~✓~~			~~✓~~	
~~Skein~~	~~✓~~			~~✓~~	~~✓~~		~~✓~~				~~✓~~	
~~SHA-512~~		~~✓~~			~~✓~~				~~✓~~		~~✓~~	
~~SHA-256~~		~~✓~~		~~✓~~			~~✓~~			~~✓~~		

~~Table x - eBASH performance comparison for long (> 4096-byte) messages * Only on processors with AES NI~~

Algorithm	AMD64			X86			ARM-NEON			32-bit RISC		
	High	Med	Low	High	Med	Low	High	Med	Low	High	Med	Low
Blake-512	✓				✓		✓				✓	
Blake-256	✓	✓		✓			✓			✓		
Grøstl-512			✓			✓			✓			✓
Grøstl-256		✓*	✓			✓			✓			✓
JH		✓			✓	✓		✓				✓
Keccak-512		✓			✓			✓			✓	
Keccak-256		✓			✓	✓		✓			✓	
Skein	✓			✓	✓			✓			✓	
SHA-512		✓				✓			✓		✓	
SHA-256		✓		✓			✓			✓		

Table y - eBASH performance comparison for 64-byte messages

* Only on processors with AES NI instructions

1.2.2 XBX: Embedded Microcontrollers

Most of our data on embedded microcontrollers comes from the XBX effort. XBX [Wenzel-Benner 12] focused on embedded computers where it is usual to compile or assemble code on some development system, and then run it on a small computer or "microcontroller." They collected and measured SHA-3 implementations on eight embedded platforms. The website for the effort is at: http://xbx.das-labor.org/trac/wiki.

Komentář [Meltem17]: I prefer a more compact table, so I combined the tables in Figure x. We may do the same for Figure y, if you prefer. I removed the colors, but we can put them back.

Naformátováno: Doleva

Komentář [Meltem18]: I prefer a more compact table, so I combined the tables in Figure x. We may do the same for Figure y, if you prefer. I removed the colors, but we can put them back.

Naformátováno: Doleva

Naformátováno: Nadpis 3

Naformátováno: Odrážky a číslování

Some of the 32-bit ARM processors benchmarked on XBX were as powerful as some of the machines tested by eBASH, and included vector units, while others were 8- and 16-bit machines characteristically used with very little memory.

In contrast to eBASH, XBX gives the RAM and ROM requirements of the fastest implementations for each processor tested, and “area” versus speed plots, for each of the implementations measured. The area metric used in the XBX is area = 4 × RAM + ROM. This is a heuristic based on the rule of thumb that RAM memory requires about 4 times the area on a chip as ROM memory. In the larger systems used in eBASH, the memory required by hash algorithms is rarely an issue, but memory use is often an issue and sometimes the major constraint with embedded microcontrollers. In addition to XBX, [Yang 12] gives optimized implementations for all five finalists plus SHA-2 on the ARM-11 processor, which is very commonly used in cell phones.

Eight-bit: Only one 8-bit microcontroller was tested, the Atmel ATmega128P. The smallest area and Grøstl-256 attained the best performance~~ was attained by Grøstl-256~~, perhaps the only software case where Grøstl was the fastest algorithm. This probably is because of the 8-bit orientation of the AES operations used by Grøstl. Blake-256 was nearly as fast as Grøstl, but took somewhat more area. Keccak-256 was also compact and third in throughput. Interestingly, SHA-256 was comparatively big and slow and SHA-512 had the worst overall performance of any algorithm.

Sixteen-bit: Only one 16-bit microcontroller, the Texas Instruments’ MSP430FG4618, was tested. Overall, Blake-256 was fast and small, and SHA-256 was second. Grøstl-256 was small but fairly slow. Keccak-256 was a bit bigger than Blake-256 and somewhat faster than Grøstl-256, but no match for either Blake-256 or SHA-256. Keccak-256 has relatively small area and reasonably good throughput.

ARM (thumb instructions): This version of the ARM processor is a low-end microcontroller that only implements the 16-bit thumb instructions. The XBX authors count small area to be the primary goal for this platform, and Blake and JH-256 are the two smallest algorithms. Blake-256 and SHA-256 have the highest (very similar) throughputs, but the minimum area requirement~~s~~ of SHA-256 ~~are ~~is higher than Blake-256. Keccak has a fairly small area requirement, and Keccak-256 is third in overall performance. Skein has the largest area requirement, and Grøstl-512 and JH are simply slow.

Komentář [SC19]: Shall we use the singular form instead of the plural form here?

32-bit RISC: This category includes several ARM processors running the regular 32-bit instructions (but without the NEON vector processor) and a 32-bit MIPS processor. Both XBX and [Yang 12] give us data here. SHA-256 is the fastest algorithm overall, followed by Blake-256. Since these are both 32-bit oriented ARX algorithms and these are all 32-bit RISC architectures, it is ~~not surprising~~expected that they should be fastest; but in general-purpose computers, Blake-256 had the edge over SHA-256. ~~We suspect that this is because ~~AMD64 and X86 machines have vector units or dual launch superscalar ALUs and apparently are more readily able to exploit the parallel processing opportunities of Blake-256, than those of SHA-256. Skein, Blake-512, Keccak-256 and SHA-512 vie for third place, depending on the specific processor, while Grøstl-256, Grøstl-512 and JH are at the bottom, much (typically 3 to 8 times) slower than SHA-256. In area, Blake-256 and Grøstl-256 are usually the smallest.

ARM with NEON: This is a relatively fast ARM core with the addition of a vector unit that supports 64-bit operations. With the addition of the vector unit, Skein becomes the fastest algorithm, followed fairly closely by Blake-256, Blake-512 and SHA-256. Keccak-256 is about half the speed of Skein while Keccak-512 about three times slower. JH and Grøstl are four to nine times slower than Skein. The overall results are similar to those on eBASH for ARMs with the NEON vector unit, although JH seems to do somewhat worse in the XBX benchmarks than the eBASH results.

The eBASH investigators give an overall ranking for embedded computers that factors in their understanding of whether the usual goal for the particular computer is throughput or minimum area. In that ranking Blake is first, while Skein barely edges out Keccak and Grøstl for second place. They did not include SHA-2 in this ranking; however, considering the good performance of SHA-256 on 32-bit RISC machines, if SHA-2 were included, it might beat Skein for second place, since there are four 32-bit platforms where SHA-2 is the fastest algorithm, although rarely by a large margin. But the smallest SHA-256 implementations nearly always require somewhat more area than the smallest Blake-256 implementations, so Blake-256 has the overall advantage.

Komentář [SC20]: I assume you meant to say "SHA-2" here.

1.2.3 The Future

Naformátováno: Nadpis 3

Naformátováno: Odrážky a číslování

An obvious starting point for projecting the future is performance data from the recent higher end processors. Figure z graphs eBASH data for three recent AMD64 architecture processors. In these graphs, each bar represents the mean cycles per byte for the fastest implementation of each algorithm on one of the three processors. Smaller bars mean faster throughput. The five finalist algorithms plus SHA-2 are plotted; however, in the case of Skein and JH, only one bar is plotted, because all four output sizes use the same compression function, and run at about the same rate. Following our nomenclature, where we used one designation for two or four message digest sizes when they run at the same rate, we plot one bar for each version with a different rate. For example, since Blake-256 and Blake-224 run at the same speed, we plot only Blake-256. But we plot all four digest sizes for Keccak, because each runs at a different speed. For consistency with other algorithms, the nomenclature for Keccak has also been changed in this report from eBASH, which generally labels Keccak variants by their capacity; for example, the eBASH "keccakc512", which outputs a 256-bit message digest, becomes "keccak-256" in Figure z.

Much of the ISA development activity in the past decade has been the addition and extension of vector units and their instructions. It is reasonable to expect vector register files to continue to grow, more instructions to be added, and to find more use in even embedded systems. It seems too big an assumption to expect that because AES has inspired additional support instructions, that SHA-3 would also do so. If it did it would likely be far down the road, unless the instructions, like vector rotates, have more general uses than implementing hash functions.

Another kind of processor that we have not considered in the competition is Graphics Processor Units (GPUs) that are common on processors for laptop, desktop and even tablet computers. These are arrays of fairly specialized processors organized for stream processing and are most commonly used for image rendering. Gaming and high definition animation have driven their development and they have been applied cryptography, but more for cryptanalysis than for protecting data; GPUs are well suited to applications that can be divided into many independent processes, such as password cracking. But it is hard for to envision much near term use for data protection.

Figure z-1 plots the cycles required per message byte of long messages for “h6sandy,” an Intel® Core i3-2310M[1], a “Sandy Bridge” internal architecture processor with two 2100 MHz cores, intended for laptop computers. Figure z-2 plots cycles per byte for “sandy0,” an Intel Core i7-2600K, a Sandy Bridge four core 3400 MHz processor intended for relatively high-end desktop computers. Figure z-3 plots cycles per byte for “hydra6,” an AMD FX-8120, a “Bulldozer” internal architecture processor with four 3100 MHz cores, also for desktop applications. The Intel Sandy Bridge processors implement the AVX vector instruction set, with sixteen 256-bit registers; and the AMD Bulldozer processors implement the XOP vector instruction set, which is similar to the AVX with some additional extensions including, in particular, the 32- and 64-bit rotate operations (which may or may not presage similar extensions to future Intel vector units). There is one significant difference between the two Sandy Bridge processors: sandy0 implements the AES-NI instructions, and h6sandy does not. The AES-NI instructions~~is~~ affect~~s~~ only the performance of Grøstl.

Komentář [SC21]: I think you got the order (of the figures) reversed.

In addition to the five finalists and SHA-2 algorithms described above, a “tree-mode,” labeled “Bblake-256” is implemented on all three processors, while two more tree-modes, Bblake-512 and Keccak-256treed2 are implemented for the AMD FX-8120. These modes generate a single 512 or 256-bit hash from two parallel streams using the regular compression function, and are examples of “Processing Multiple Buffers in Parallel” or “multi-buffering” to facilitate efficient use of [Gopal 10a]; in these particular cases by dividing a single file into two parts to be hashed in parallel. The same technique can also be applied to two independent hash streams, as might occur on a server with many simultaneously active TLS connections.

Skein does not seem to benefit from the vector units, largely, we believe, because of the different rotation constants used in its Mix operations, even though four parallel Mix operations occur in each round. Skein, however, is one of the fastest algorithms on all three of these machines, relying primarily on their superscalar general purpose ALUs.

[1] The use of the corporate names Intel and AMD and the identification of specific product models, processor architectures and their features is to enable readers to understand and check the performance data presented here. It is not an endorsement or recommendation by NIST of any company, product or feature.

Blake logically allows parallel execution of four of its building-block G-functions, with their rotates; unlike Skein, Blake's four parallel rotates all use the same rotation values, facilitating vectorization. We see that on these machines with the most advanced vector units, Blake-256, which uses 32-bit words, becomes almost as fast as Skein-256, which uses 64-bit words and in older AMD64 machines has a bigger advantage over Blake-256. Blake 512 pulls a bit ahead of Skein-512 on these machines, although Skein-512 has the advantage on older AMD64 machines.

We also see that Keccak does relatively better on the Bulldozer because only the Bulldozer implements vector rotation instructions, giving about a factor of 1.7 speed improvement [van Assche 12]. Blake also benefits from the rotate instruction [Neves 12]. A single Skein thread probably won't be helped very much by a simple rotate instruction that rotates all words by the same amount. Grøstl is significantly helped by the AES-NI instructions and corresponding AES extension on the Bulldozer. JH clearly benefits from vector implementations, but still is among the slowest on all three processors. ~~It is hard to rule out the possibility that there is some better way to vectorize JH that makes it faster relative to the other units, but it is not obvious what it would be.~~ Neither JH nor Grøstl seem to benefit from a rotate instruction.

All of the candidates as well as SHA-2 potentially can benefit from tree mode or multi-buffer implementations, although perhaps to varying degrees. The different rotation values that complicate vectorizing a single Skein thread should not matter in a multi-buffer implementation. But there has not been enough work done on this to draw strong conclusions about which algorithms benefit most.

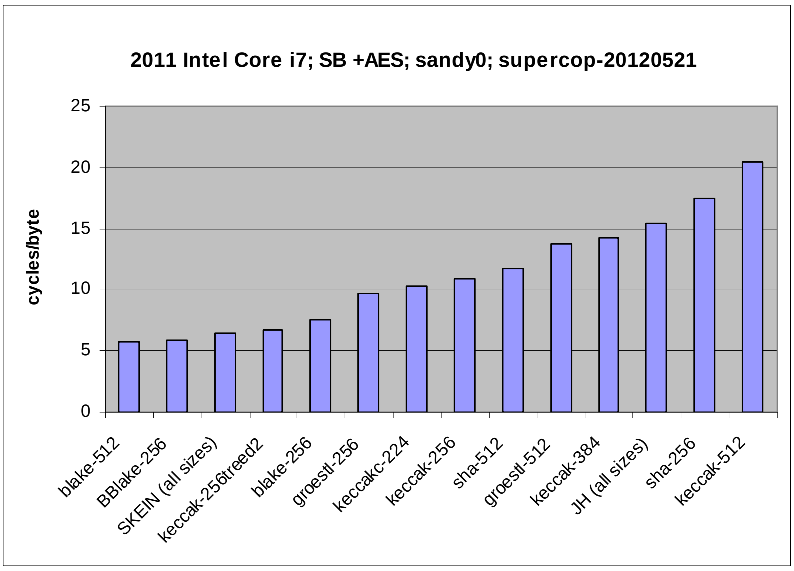

Figure z-1 – SHA-3 Finalist Cycles/Byte on Current Sandy Bridge Desktop Processor

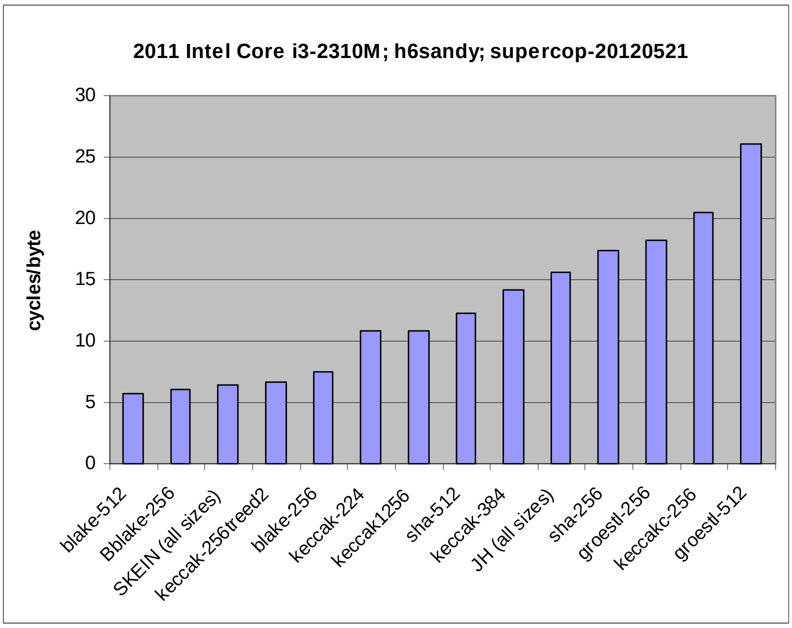

Figure z-2 – SHA-3 Finalist Cycles/Byte on Current Sandy Bridge Laptop Processor

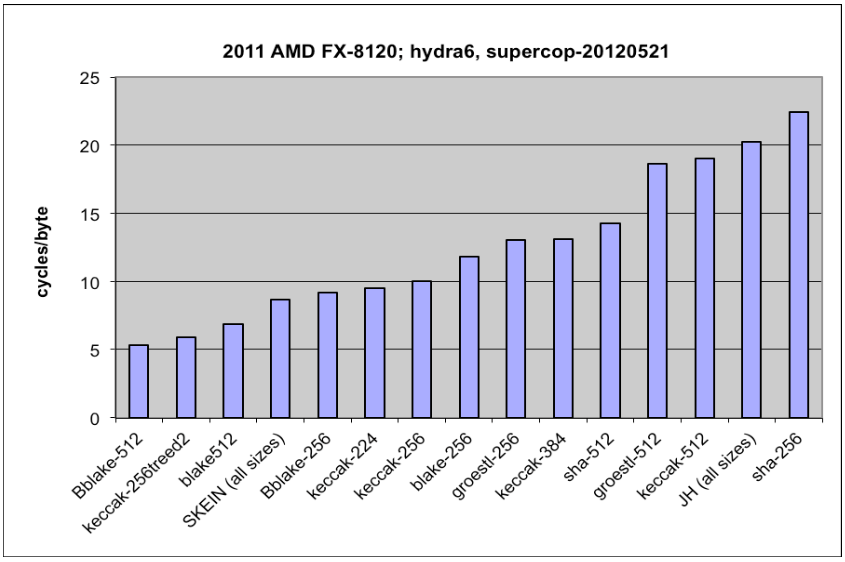

Figure z-3 - SHA-3 Finalist Cycles/Byte on Current Bulldozer Desktop Processor

1.2.4 Software Performance Summary

Naformátováno: Nadpis 3

Naformátováno: Odrážky a číslování

Skein and Blake, the two ARX finalist candidates, have the best overall software performance. Only Skein and Blake seem to be faster than SHA-2 in most cases.

Skein has the advantage of a single compression function for all four message digest sizes, which saves memory if all four sizes are required. Skein-256 has a small to moderate performance advantage over Blake-256 on AMD64 platforms, which may, or may not carry over to future 64-bit ARMv8 processors, if ~~they~~ the processors adopt super-scalar ALUs similar to the ones used in AMD64 processors. Blake-512 seems to gain a modest performance advantage over Skein-512 on more recent AMD64 machines by better fitting their vector units.

Komentář [Meltem22]: Who?

On 32-bit machines (mainly ARM processors) without vector units, Blake-256 is the clear overall leader, although it does not offer any real speed advantage over SHA-256. On ARMs with the NEON vector unit, Skein seems the fastest algorithm, followed fairly closely by Blake.

Naformátováno: Zarovnat do bloku

On small embedded computers, Blake-32 has decisively the best overall performance. Blake-32's maximum throughput is often similar to, or sometimes less than SHA-256, but Blake-32 generally had smaller memory requirements than SHA-2 and most of the other candidates.

Komentář [SC23]: Is this data prior to round3? The name indicates so.

Naformátováno: Zarovnat do bloku

Keccak is reasonably fast for smaller digest sizes on a range of machines, but Keccak-512 is among the slowe~~r~~st algorithms on most machines. Keccak benefits significantly from vector rotate instructions, which may become common on future processors. Like Skein, Keccak uses one compression function for all four digest sizes, and is amenable to compact implementations.

Naformátováno: Zarovnat do bloku

Grøstl-256 seems to need hardware support of AES S-boxes to achieve even the medium level of performance on computers with a 64- or 32-bit word orientation. Even with such support, Grøstl ~~Grostel~~-512 ranks near the bottom. ~~–~~However, on the one 8-bit microcontroller tested, Grøstl-128 was the fastest ~~candidate~~ finalist and one of the most compact.

Naformátováno: Zarovnat do bloku

JH seems to benefit from larger vector sizes, but even with vector instruction sets that allow 128-bit operations, it strains to reach even the medium performance level.

Naformátováno: Zarovnat do bloku

References

[FIPS 180] National Institute of Standards and Technology, *Secure Hash Standard*, Federal Information Standards Publication, FIPS-180, May 1993

[FIPS 180-1] National Institute of Standards and Technology, *Secure Hash Standard*, Federal Information Standards Publication, FIPS-180-1, April 1995

[FIPS 180-4] National Institute of Standards and Technology, *Secure Hash Standard*, Federal Information Standards Publication, FIPS-180-1, March 2012

[SP800-38D] Morris, Dworkin, NIST Special Publication 800-38D, NIST Special Publication 800-38D, November 2007

[Grisen 11] Richard Grisenthwaite, "ARMv8 Technology Preview," Presentation at ARM 2011 TechCon, available at: http://www.arm.com/files/downloads/ARMv8_Architecture.pdf

[Bernstein 12] Daniel J. Bernstein and Tanja Lange2 "The new SHA-3 software shootout," The Third SHA-3 Candidate Conference, March 23, 2012, available at: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/March2012/documents/papers/BERNSTEIN_paper.pdf

[Bertoni 09] Guido Bertoni, Joan Daemen, Tony Dusenge, Gilles Van Assche: Sufficient conditions for sound tree and sequential hashing modes. IACR Cryptology ePrint Archive 2009: 210 (2009). http://sponge.noekeon.org/TreeHashing.pdf

[Black97] Black, J., Halevi, S., Krawczyk, H., Krovetz, T., and Rogaway, P. UMAC: Fast and secure message authentication. In *Advances in Cryptology* {CRYPTO '99 (1999), vol. 1666 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 216-233

[Flynn72] Flynn, M. (1972). "Some Computer Organizations and Their Effectiveness". IEEE Trans. Comput. C-21: 948

[Gopal 10] Vinodh Gopal, Wajdi Feghali, Jim Guilford, Erdinc Ozturk, Gil Wolrich, Martin Dixon, Max Locktyukhin, and Maxim Perminov, "Fast Cryptographic computation on IA processors via Function Stitching," Intel Corp. White Paper, http://download.intel.com/design/intarch/PAPERS/323686.pdf

[Gopal 10a] Vinodh Gopal, Jim Guilford, Wajdi Feghali, Erdinc Ozturk, Gil Wolrich, and Martin Dixon, "Processing Multiple Buffers in Parallel to In crease Performance on Intel Architecture Processors," Intel Corp. White Paper, July 2010, http://download.intel.com/design/intarch/papers/324101.pdf.

[Gopal 11a] Vinodh Gopal, Jim Guilford, Wajdi Feghal, Erdinc Ozturk, Gil Wolrich, Kirk Yap, Sean Gulley, and Martin Dixon, "Cryptographic Performance on the 2nd

Generation Intel Core Processor," Intel Corp White Paper, January 2011, http://download.intel.com/design/intarch/PAPERS/324952.pdf

[Gopal 11b] Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Sean Gulley, and Wajdi Feghali "Improving OpenSSL Performance," Intel Corp White Paper, October 2011, http://download.intel.com/design/intarch/papers/326232.pdf

[Gueron 12] Shay Gueron and Vlad Krasnov "Parallelizing message schedules to accelerate the computations of hash functions," February, 2012, http://eprint.iacr.org/2012/067.pdf

[Kelsey06] John Kelsey, Tadayoshi Kohno - Herding Hash Functions and the Nostradamus Attack, In Proceedings of EUROCRYPT, LNCS 4004, pp. 183-200, Springer, 2006

[Kelsey05] John Kelsey, Bruce Schneier - Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work, In Proceedings of EUROCRYPT, LNCS 3494, pp. 474-490, Springer, 2005

[Neves 12] Samuel Neves and Jean-Paul Aumasson, "Blake and 256-bit advanced Vector Extensions," The Third SHA-3 Candidate Conference, March 2012, available at: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/March2012/documents/papers/NEVES_paper.pdf . Presentation available at: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/March2012/documents/presentations/NEVES_presentation.pdf

[Stevens 07] Marc Stevens and Arjen Lenstra and Benne de Weger, "Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities"
In: Proceedings of Advances in Cryptology - EUROCRYPT 2007, Barcelona, Spain, May 2007, Springer Verlag LNCS, vol. 4515, pp. 1-22.

[Wang05] Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu' "Finding Collisions in the Full SHA-1", Crypto 2005

[Wenzel-Benner 12] Christian Wenzel-Benner, Jens Gräf, John Pham, and Jens-Peter Kaps, "XBX Benchmarking Results January 2012," The Third SHA-3 Candidate Conference, March 23, 2012, available at: http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/March2012/documents/papers/WENZEL_BENNER_paper.pdf

[van Assche 12] Gilles Van Assche, "Keccak, tree hashing and rotation instructions" e-mail to Hash Forum, available at: http://csrc.nist.gov/groups/ST/hash/email_list.html, May 30, 2012

Acknowledgements

eBASH was supported by the Virtual Applications and Implementations Research Lab (VAMPIRE), a part of the ECRYPT II project. Tanja Lange, of the University of Eindhoven, and Daniel Bernstein, of the University of Illinois, Chicago, organized and led eBASH, which was the third such cryptographic benchmarking effort organized under VAMPIRE to measure the performance of cryptographic algorithms.

The XBX effort was organized for the SHA-3 competition by Christian Wetzel-Benner and Jens Gräf

Naformátováno: Písmo: Tučné

Appendix A – e-Bash shootout Plots

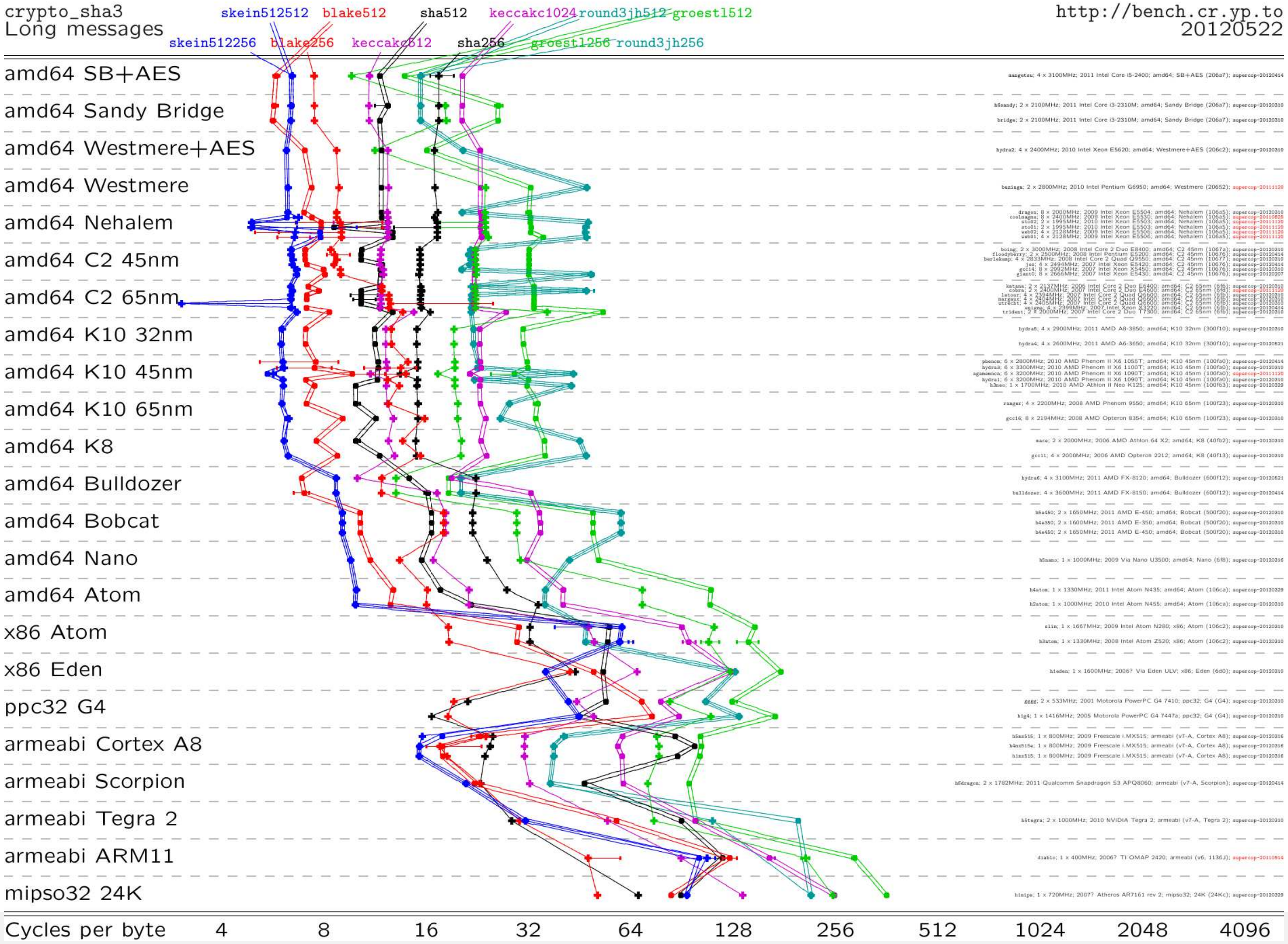

Figure A-1 – eBash SHA-3 Finalist Performance for Long Messages

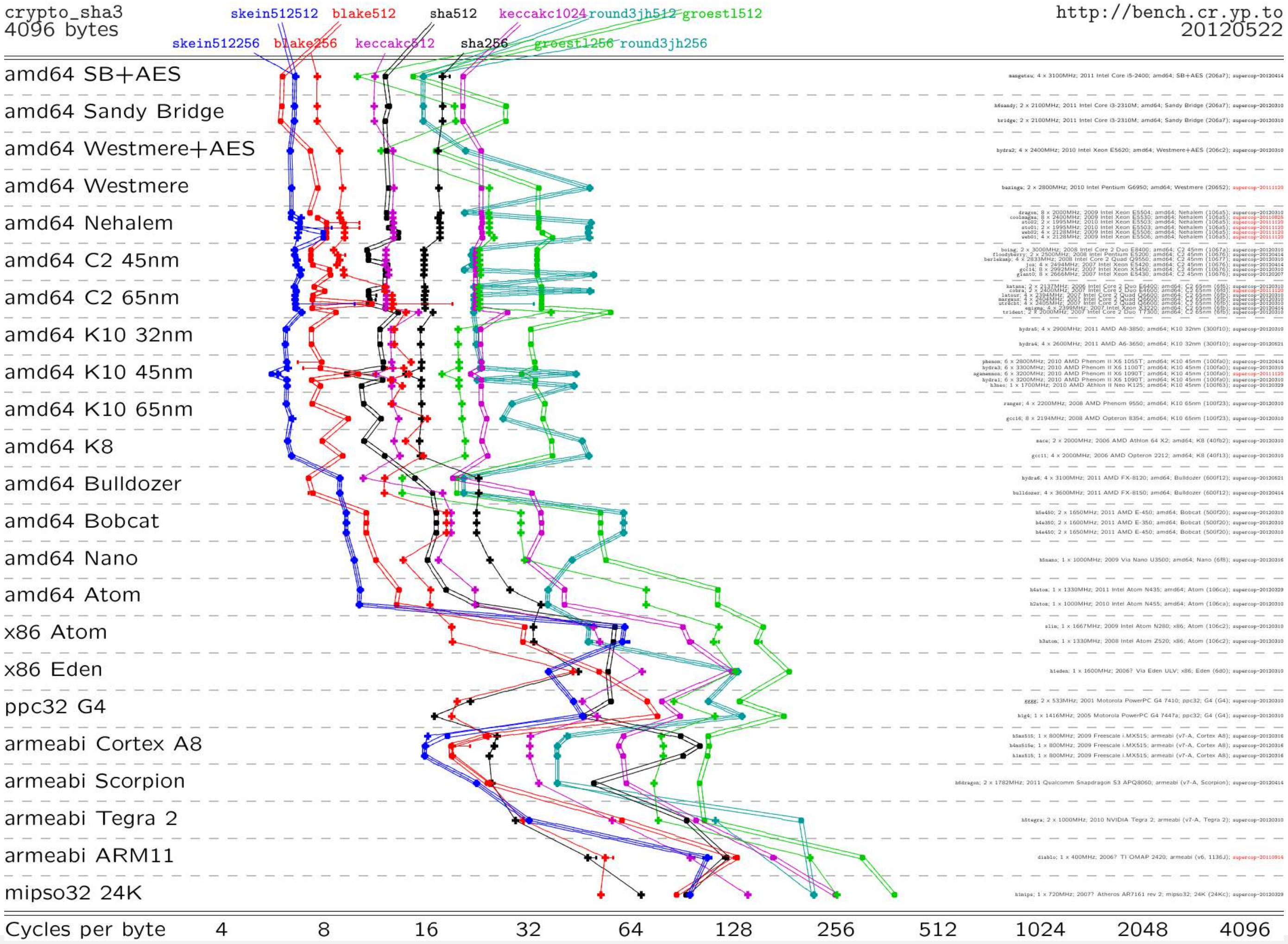

Figure A-2 – eBash SHA-3 Finalist Performance for 4096 Byte Messages

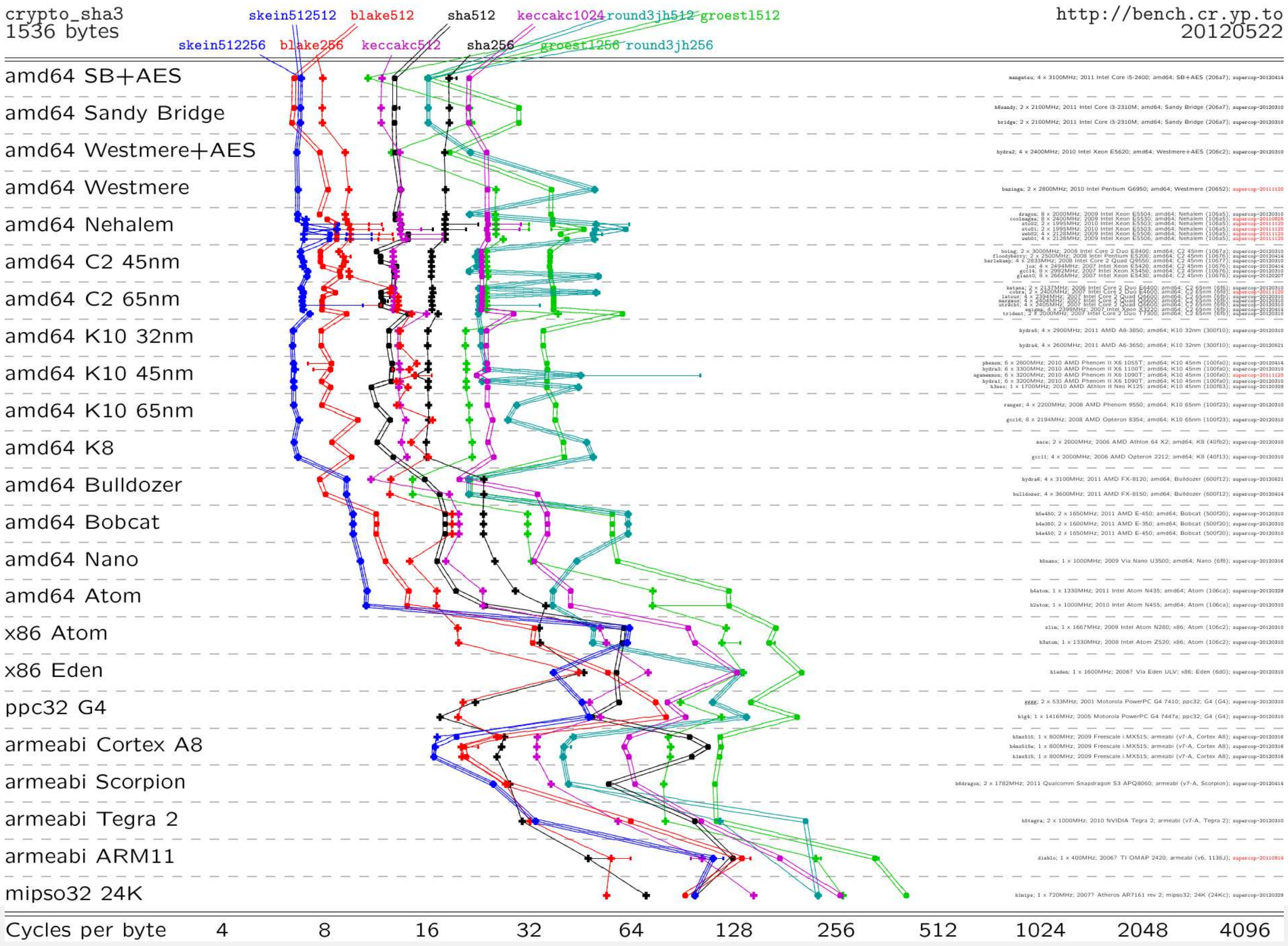

Figure A-3 – eBash SHA-3 Finalist Performance for 1536 Byte Messages

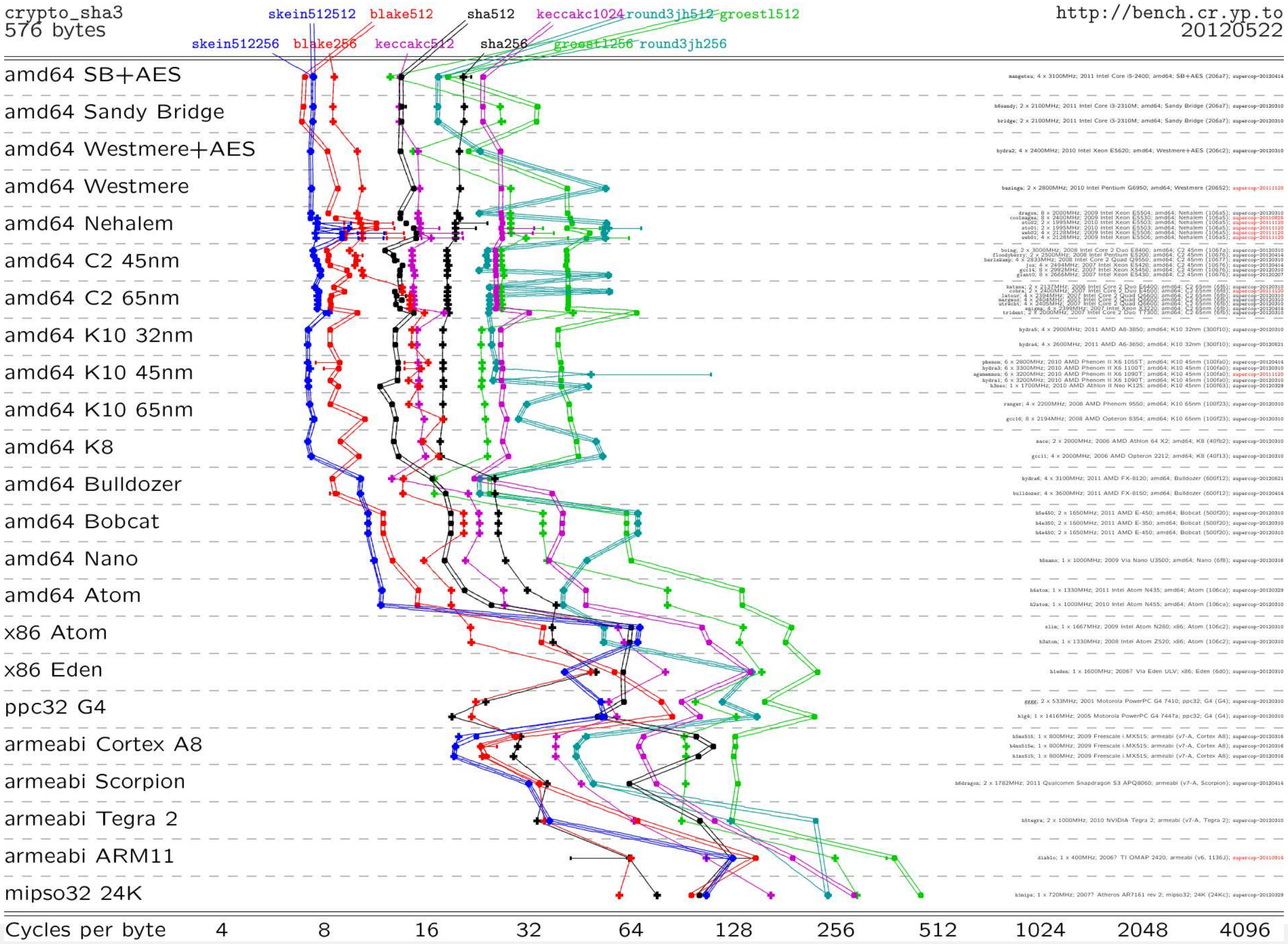

Figure A-4 – eBash SHA-3 Finalist Performance for 576 Byte Messages

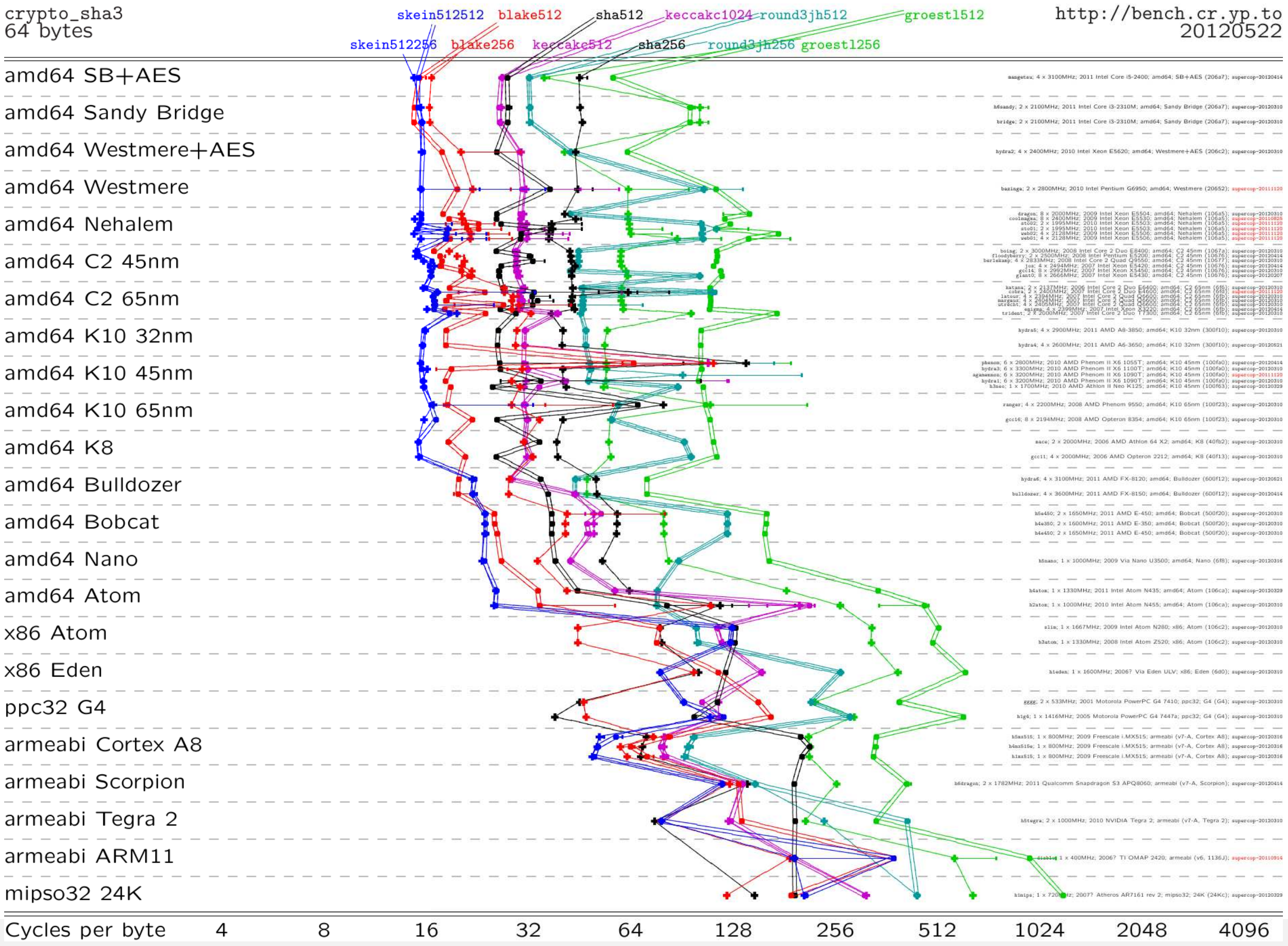

Figure A-5 – eBash SHA-3 Finalist Performance for 64 Byte Messages

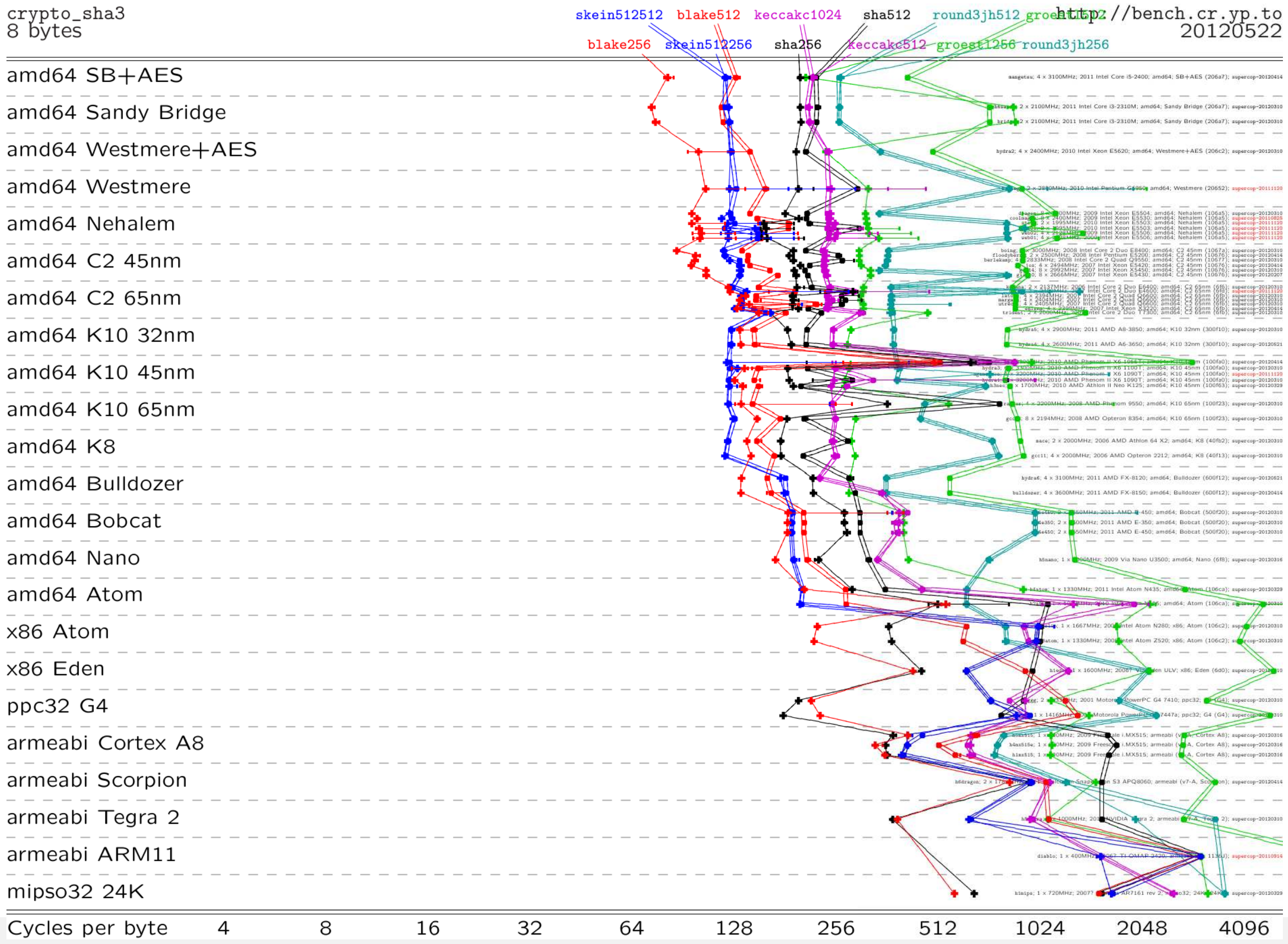

Figure A-6 – eBash SHA-3 Finalist Performance for 8 Byte Messages