CubeHash parameter tweak: 10× smaller MAC overhead

Daniel J. Bernstein *

Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607–7045
cubehash@box.cr.yp.to

1 Introduction

CubeHash$i$+r/b+f–h performs i initialization rounds, r rounds for each b bytes of input, and f finalization rounds. It then produces h bits of output.

The first-round CubeHash submission (October 2008) identified r and b as tweakable security parameters. The time taken by CubeHash for a long message is approximately proportional to r/b. My original recommendations for (r, b) drew criticism as making CubeHash too slow.

The second-round CubeHash submission (September 2009) made new recommendations for (r, b) and identified i and f as additional tweakable security parameters. CubeHash has a per-message initialization overhead approximately proportional to i, although this is skipped by implementations that precompute the initialization. CubeHash also has a per-message finalization overhead approximately proportional to f; this cannot be skipped. My recommendations for (i, f) have drawn criticism as making CubeHash too slow for short messages.

The most common argument for worrying about short-message hashing speed is that it is critical for packet authentication. Consider, for example, a 1500-byte Internet packet hashed by HMAC-CubeHash. Finalization in the second-round CubeHash submission is comparable to hashing 320 bytes. HMAC hashes twice, incurring an *extra* finalization, comparable to hashing another 320 bytes. Overall the CubeHash finalization adds 640 bytes to the original 1500, a 43% overhead. For smaller packets these 640 bytes are even more important.

Contents of this document. This document covers three issues:

- The cost of finalization in CubeHash. This document proposes reducing the number of finalization rounds from 160 to 32, reducing the finalization overhead by a factor of 5.

* The author was supported by the National Science Foundation under grant ITR–0716498. Date of this document: 2010.10.31.

- The number of finalizations involved in a MAC using CubeHash. This document proposes switching from HMAC to a classic prefix MAC, reducing the MAC overhead by another factor of 2.
- The number of CubeHash options proposed for SHA-3. This document makes a new list of specific CubeHash options that I propose for SHA-3. This document also includes a table of security levels achieved by each of these options.

All of this is discussed in detail below.

2 Analyzing security margins

When Rijndael–256 was chosen as AES–256, it needed 10 rounds to resist all attacks that were known at the time—but it had 14 rounds (40% extra computation) as a security margin. Serpent–256, widely regarded as the second-place candidate, needed 10 rounds to resist all attacks that were known at the time—but it had 32 rounds (220% extra computation) as a security margin. The Serpent designers deliberately prioritized security, leaving a very comfortable security margin, but were punished in evaluations of software speed.

One can see similar variation in the SHA-3 candidates. Here are three examples:

- CubeHash8/64–256 has 0.125 rounds per input byte and resists all attacks known. My second-round proposal CubeHash16/32–256 has 0.5 rounds per input byte (300% extra computation) as a security margin.
- 5 rounds of ECHO-256 resist all attacks known. ECHO-256 has 8 rounds (60% extra computation) as a security margin.
- 7 rounds of Grøstl–256 resist all attacks known. Grøstl–256 has 10 rounds (43% extra computation) as a security margin.

The best relevant results are collision attacks estimated to take 2^{71} simple operations against CubeHash5/64–256; 2^{132} simple operations against CubeHash6/64–512; 2^{80} simple operations against CubeHash8/96–256; 2^{64} simple operations against 4 rounds of ECHO-256; and 2^{112} simple operations against 6 rounds of Grøstl–256.

The Rijndael-vs.-Serpent example suggests that the community does not place a large value upon a large security margin; perhaps this is why many SHA-3 candidates have rather small security margins. Another, more recent, example is the selection of 12-round Salsa20/12 (50% extra computation) as part of the eSTREAM portfolio, despite the designer's recommendation to use 20-round Salsa20 (150% extra computation).

Arguments for a large security margin in SHA-3. Here are three arguments that SHA-3 should have a large security margin. First, hash-function speed is important to far fewer users than cipher speed; in almost all applications of SHA-3, something *else* will be the bottleneck. My model of the future SHA-3 users is as follows:

- 1% of the users will need SHA-3 to be fast.

- 10% of the users will need SHA-3 to be small.
- 100% of the users will need SHA-3 to be secure.

In particular, users do *not* need HMAC to be fast, even in applications where message authentication is a bottleneck: for years there have been extremely fast MACs that do *not* rely on hash functions and that are guaranteed to be as secure as the user's favorite cipher. (I am not saying that fast HMAC is undesirable!)

Second, hash-function security is a considerably more complicated concept than cipher security. There are many different hash-function applications, with several different notions of security—increasing the risk that cryptanalysts evaluating SHA-3 proposals will miss an attack relevant to some protocols and not others, and increasing the importance of a large security margin.

Third, even in the cipher context, I think that history has shown that the Serpent designers were right. Here is what they wrote in "The case for Serpent" in 2000:

> Ciphers can also be damaged through erosion of public confidence ... Differential and linear attacks became translated in the public mind to "DES has been broken". ... Having experienced what happened to DES, we are concerned that, in perhaps 50 years' time, advances in mathematics will lead to a certificational attack on 16-round Serpent. As the other AES finalists have no more margin of safety than 16-round Serpent, they run a similar risk.

I think it's clear *to cryptographers* that the recent related-key "attacks" on AES will never be of any use for any real-world attacker, but the "attacks" have nevertheless reduced public confidence in AES. A larger security margin would have avoided this trouble.

3 The security margin for CubeHash finalization

The second-round CubeHash proposal is

- CubeHash160+16/32+160–224 for SHA-3-224,
- CubeHash160+16/32+160–256 for SHA-3-256,
- CubeHash160+16/32+160–384 for SHA-3-384–normal,
- CubeHash160+16/32+160–512 for SHA-3-512–normal,
- CubeHash160+16/1+160–384 for SHA-3-384–formal, and
- CubeHash160+16/1+160–512 for SHA-3-512–formal,

The 160-round finalization is (except for the "formal" proposals) comparable to hashing 320 bytes, as mentioned above. This finalization is obviously massive overkill, and has generated complaints, so it should be reduced.

There is an argument that finalization isn't necessary at all. If an r-round transformation is "indistinguishable from a random oracle" (or "hermetic" or some other equivalent notion

that makes theoreticians cringe) then the same r rounds can be used for each block, including the last block. The last block has to be tagged somehow before it is processed, and the output has to be significantly smaller than the total RAM use, but no further protection is required.

There is a counterargument saying that finalization adds a valuable security margin against first-preimage attacks. The last message block is transformed through r rounds *plus* f finalization rounds, forcing the cryptanalyst to work backwards through many more computations. Of course, replacing (r, f) by $(r + f, 0)$ would achieve the same last-block security, but it would be much more expensive in all previous blocks. Several SHA-3 candidates have $f > 0$: CubeHash, Fugue, Hamsi, Luffa, Shabal, and SIMD.

If finalization is very strong then it drastically limits the ways in which the cryptanalyst can attack the rest of the hash function. As an extreme example, if finalization is "hermetic" then the only job of previous message processing is to avoid internal collisions. (If finalization is "hermetic" modulo some symmetries then the only job of previous message processing is to avoid internal collisions modulo the same symmetries.) If the attacker is unable to create internal collisions then the most that he can hope to achieve is to create distinct finalization inputs with some sort of structure; if the finalization is strong then it will destroy this structure.

It seems to me that a CubeHash round has, overall, similar strength to a Serpent round. (This assessment is based on extensive third-party analysis of differential attacks on reduced-round CubeHash. Recent linear attacks make a CubeHash round seem even stronger than a Serpent round.) I expect 16 finalization rounds to be secure for every standard application, and in particular for the PRF/MAC applications discussed in more detail below. I therefore recommend 32 finalization rounds, providing a very comfortable security margin. I also recommend 16 initialization rounds.

4 CubeMAC

The standard approach to constructing a secure MAC is to construct a "PRF"; i.e., a secure cipher; i.e., a cipher whose outputs for distinct inputs are indistinguishable from uniform; i.e., a cipher whose outputs for distinct inputs are indistinguishable from independent uniform random strings. For example, AES with a secret 256-bit key is believed to be indistinguishable from uniform when the number of inputs is significantly below 2^{64}. ("Related-key attacks" are irrelevant to the PRF concept.)

The obvious problem with this approach is that typical ciphers take only short fixed-length inputs: for example, AES takes only a 128-bit input. The standard solution is to start with a compression layer that maps long variable-length inputs to short fixed-length outputs. There are extremely fast "universal hash functions" that use a secret key and that are guaranteed to be information-theoretically secure in this context: if all relevant sizes are chosen sensibly then the attacker has negligible probability of finding two colliding

messages. It is easy to prove that the resulting MAC is indistinguishable from uniform, and therefore a secure MAC, if the original cipher is indistinguishable from uniform.

HMAC and NMAC follow this approach, using a cryptographic hash function such as SHA-1 for both layers. A message m is first hashed to $h = \text{SHA-1}(k_1, m)$, and then the hash is encrypted to $\text{SHA-1}(k_2, h)$. It is easy to prove that $\text{SHA-1}(k_2, \text{SHA-1}(k_1, m))$ is indistinguishable from uniform, and therefore a secure MAC, if two hypotheses hold: first, $\text{SHA-1}(k_1, m)$ has low collision probability; second, $\text{SHA-1}(k_2, h)$ is indistinguishable from uniform on 160-bit inputs h. Neither of these hypotheses has been disproven, and if SHA-1 is replaced by something stronger then both hypotheses become very easy to believe.

Why not simply use $\text{SHA-1}(k_1, m)$ as a MAC? The answer is that SHA-1 is vulnerable to trivial length-extension attacks: the structure of SHA-1 means that $\text{SHA-1}(k_1, m)$ reveals $\text{SHA-1}(k_1, m')$ for many extensions m' of m. But all of the SHA-3 candidates avoid this problem. For wide-pipe candidates with very strong finalization—in particular, for CubeHash—it is clear that the extra work in NMAC and HMAC does not accomplish anything. The state before finalization determines the key that was used (in the absence of internal collisions), so re-injecting the key is redundant; finalization then hides the state as effectively as a subsequent hash would.

I therefore propose CubeMAC: put a 512-bit secret key in front of the input message, feed the result to CubeHash, and use the output directly as a MAC. The overhead for this MAC is the overhead for a single finalization, comparable to hashing just 64 bytes—ten times smaller than the second-round HMAC-CubeHash proposal. For short messages the total cost of computing the MAC is comparable to hashing 96 bytes, if the CubeHash state after the secret key is precomputed.

I don't claim that CubeMAC is competitive in performance with non-hash-based MACs. However, it is faster, often much faster, than HMAC-SHA-256 and HMAC-SHA-512 for all message lengths across a huge range of software and hardware platforms. CubeMAC also provides exceptionally low hardware complexity for a combined high-security hash-and-MAC unit.

5 Impact of tweaks upon existing analysis

Some SHA-3 candidates have advertised the fact that they haven't been tweaked. This is not an unreasonable advertisement—I think that tweaks can cause some big problems. However, these problems aren't applicable to the CubeHash tweaks.

The main problem with tweaks is that they burn cryptanalytic time, the most valuable resource in the SHA-3 competition. Often a tweak responds directly to cryptanalytic results, with the explicit goal of undermining those results; some examples are

- the existing tweak to BMW (to undermine some free-start "attacks"),
- the announced tweak to Grøstl (to undermine some reduced-round collision attacks),

- the existing tweak to SHAvite-3 (to undermine some free-start chosen-counter chosen-salt "attacks"),
- the announced tweak to SHAvite-3 (same), and
- the announced tweak to Skein (to undermine some reduced-round related-key "attacks").

Even if a tweak is not *intended* to undermine any particular result, it forces cryptanalysts to spend time carefully reviewing previous work to see whether the work has been affected by the tweak.

This problem does not occur for tweaks that merely change recommendations for security parameters: for example, the existing tweak to CubeHash (extra rounds, larger blocks), the existing tweak to Keccak (extra rounds, larger blocks), and the new tweak to CubeHash (reduced finalization). Cryptanalysts were already free to adjust the security parameters; whatever analysis they did, with whatever security parameters they chose, remains perfectly valid analysis of the tweaked function (assuming it was valid to begin with)—because the function didn't actually change! In particular:

- All of the larger-block-size analysis of CubeHash remains perfectly valid larger-block-size analysis of CubeHash.
- All of the reduced-round analysis of CubeHash remains perfectly valid reduced-round analysis of CubeHash.
- All of the reduced-finalization analysis of CubeHash remains perfectly valid reduced-finalization analysis of CubeHash.

There is no need to spend time checking anything.

Of course, these tweaks *do* require new benchmarking. However, benchmarking time is completely different from cryptanalytic time; benchmarking time is not a precious resource that needs to be spent carefully.

6 Analyzing the number of options in SHA-3

"Be specific about which variants you are proposing," NIST stated at the Second SHA-3 Candidate Workshop. "Fewer variants is better."

Every SHA-3 proposal is required to include *at least* four different options: an option providing 224-bit output (with 2^{224} security), an option providing 256-bit output (with 2^{256} security), an option providing 384-bit output (with 2^{384} security), and an option providing 512-bit output (with 2^{512} security).

For the majority of the SHA-3 proposals there are more than four options—often many more. This section discusses the core arguments for and against multiple options.

Flexibility. The core argument for additional options is that additional flexibility will be useful for the SHA-3 users. NIST already encouraged flexibility in its call for submissions:

> Candidate algorithms with greater flexibility will meet the needs of more users than less flexible algorithms, and therefore, are preferable. However, some extremes of functionality are of little practical use (e.g., extremely short message digest lengths)—for those cases, preference will not be given.

NIST gave three examples of possible types of flexibility. Two of the examples are of *implementation* flexibility for a single hash function, but one of the examples means having a wider range of hash functions: "The algorithm has a tunable parameter which allows the selection of a range of possible security/performance tradeoffs."

The Keccak SHA-3 proposal specifies two different hash functions with 512-bit output: Keccak[c=1024]-512 and the "default" Keccak[]-512. There is a tradeoff between security and efficiency: Keccak[]-512 is faster, and is the topic of practically all of the Keccak speed advertisements, but limits the user to "only" 2^{288} preimage security.

There are also two different 512-bit CubeHash options, two different 512-bit ECHO options (ECHO-DP and ECHO-SP), two different 512-bit Fugue options (after an upcoming tweak announced by the designer), and two different 512-bit Skein options (Skein-512-512 and Skein-1024-512). In each case the user is given the choice between a lower-performance higher-security 512-bit option and a higher-performance lower-security 512-bit option.

There are other types of flexibility. ECHO advertises as an advantage its "smooth support—using the same implementation—of any hash output of length from 128 to 512 bits." Skein similarly says that its support for "any output size" simplifies "many applications." Luffa says that its design "allows to generate bit strings of arbitrary length by iterating the output function OF and the round function Round. This feature is useful for some applications."

BLAKE provides another type of option: a "built-in salt." It explicitly says that providing an "interface for hashing with a salt" is an advantage and that this option "simplifies a lot of things; it provides an interface for an extra input, avoids insecure homemade modes, and encourages the use of randomized hashing." SHAvite-3 and Skein have similar options. Skein and Keccak also offer encryption options, tree options, etc., in each case arguing that this flexibility will be helpful for some SHA-3 users.

Interoperability. Supporting many hash options usually requires much more space—RAM, code size, hardware area, etc.—than supporting just a few options. This poses serious problems for platforms that have hard space limits: RFIDs, for example, and tiny microcontrollers. It also poses serious problems for typical embedded "system on chip" applications: hashing has to compete with many other functions for chip area.

Hardware designers often react to this problem by implementing only the options, or single option, that they need today. For example, if a protocol today needs SHA-256, hardware designers are not likely to burn both SHA-256 and SHA-512 into a chip supporting the protocol; burning just SHA-256 into the chip costs far less area. Upgrading the protocol to support SHA-512 then requires an extremely expensive hardware upgrade.

These issues are discussed in more detail in my paper "SHA-3 interoperability." NIST's call for submissions already recognized the value of interoperability—

> For interoperability, NIST strongly desires a single hash algorithm family (that is, that different size message digests be internally generated in as similar a manner as possible) to be selected for SHA3.

—although it did not identify unified-option hardware area as a way of quantifying this value.

Does this interoperability argument mean that a function with a smaller number of options is better than a function with a larger number of options? Not necessarily. Perhaps all of the options for the second function can be implemented together in less hardware area than the first function—making the second function better from this perspective.

Avoiding bait and switch. SuperHash is much more efficient than BoringHash: look at these benchmarks showing how small and fast SuperHash-256 is compared to BoringHash! SuperHash is also much stronger than BoringHash: look at how many rounds SuperHash-512 has, and how massive the internal SuperHash-512 pipes are, clearly much more confidence-inspiring than BoringHash!

But maybe SuperHash-256 is horrifyingly insecure, while SuperHash-512 is so inefficient as to be practically unimplementable, making *all* users unhappy with SuperHash. Maybe BoringHash is reasonably secure and efficient, making practically all the users happy.

One can view this hypothetical SuperHash example as an argument to minimize the number of options. Allowing *many* options allows easy bait-and-switch advertising: the designer creates one function with high security, one function that runs quickly in software, one function that fits into a few FPGA slices, etc., and bundles these different functions together into a single SuperHash package. Having a *single* option eliminates any possibility of this deception.

On the other hand, maybe SuperHash really is better than BoringHash. Maybe the efficiency of SuperHash-256 is accompanied by a perfectly acceptable security level, while the security of SuperHash-512 is accompanied by perfectly acceptable efficiency. Perhaps SuperHash is providing two attractive options for the users, where *each* option is better than anything BoringHash can provide.

It seems to me that the right way to analyze the merits of a SHA-3 proposal—whether the proposal is a small collection of hash functions or a large collection of hash functions—is to ask how happy the users will be with the proposal. Given this SHA-3 proposal, Alice will look for the option that best meets Alice's requirements; the first question is what Alice's requirements are (security, throughput, area, interoperability, etc.), the second question is which option will best meet those requirements, and the third question is how this compares to other SHA-3 proposals. Bob will look for the SHA-3 option that best meets Bob's requirements, raising the same set of questions for Bob. And so on.

Avoiding complexity of cryptanalysis. SHAvite-3's salt turned out to have less diffusion than desired, allowing a series of free-start chosen-counter chosen-salt "attacks" that would not have been possible if the salt option had been omitted.

I don't think this should be counted against SHAvite-3: I don't see how any sane protocol could allow these "attacks," and I don't think it's fair to punish SHAvite-3 for a salt problem when most SHA-3 submissions don't support salts at all. However, this example illustrates the principle that proposals with many options usually have more opportunities for security problems than proposals with just a few options.

Even if no security problems are found, proposals with many options usually consume more cryptanalytic time than proposals with just a few options. The diffusion of cryptanalytic effort reduces public confidence in these proposals.

On the other hand, the number of options is merely one contributor to cryptanalytic time. BMW, for example, has very few options but is widely reported as requiring tremendous effort for cryptanalysts to understand and analyze.

7 The number of options in CubeHash

This document proposes a new list of exactly 8 CubeHash options. This section discusses the choice of options, and analyzes the merits of the options according to the criteria discussed in the previous section.

Interoperability. CubeHash was designed to fit all options—including the first-round options, the second-round options, and the new list of options proposed here—into one very small circuit, avoiding interoperability problems and minimizing hardware costs. The area required for supporting all options is practically identical to the area required for a single option. Interoperability is therefore not an argument against the number of CubeHash options.

There are four other SHA-3 candidates (ECHO, JH, Keccak, and Shabal) that allow all options to fit naturally together into the same circuit, although the circuit is not as small as CubeHash. There is one other SHA-3 candidate (Skein) that has part of the same feature: Skein-512-256 and Skein-512-512 fit naturally together into the same circuit, but Skein-256-256 and Skein-512-512 do not. The other eight SHA-3 candidates handle extra options using extra hardware.

Flexible output sizes: 6 options. The minimum possible number of options allowed by the SHA-3 rules is 4, to match the 4 different output sizes allowed by SHA-2: "the submitted algorithms for SHA3 must provide message digests of 224, 256, 384 and 512 bits to allow substitution for the SHA2 family."

Some SHA-3 teams have argued that 160-bit and 128-bit options will be useful as drop-in replacements for SHA-1 and MD5. I agree. This document proposes 160-bit and 128-bit options for CubeHash, supplementing the existing proposals of 224-bit, 256-bit, 384-bit,

and 512-bit options. I see far more value in the 160-bit and 128-bit options than in the 224-bit and 384-bit options!

The underlying facts are that (1) SHA-1 and MD5 remain very widely used in a variety of existing applications and (2) many of those applications randomize their hash inputs, making 160-bit hashes quite safe until quantum computers are built, and even making 128-bit hashes reasonably safe. NIST stated in its call for submissions that substituting for SHA-1 "is not contemplated" since 160-bit hashes are "becoming too small to use for digital signatures"; but this statement takes an overly narrow view of the applications of hash functions.

Insane security levels: 1 extra option. Of course, cryptographers interested in long-term security do not recommend 128-bit or 160-bit hashes, and often do not recommend 224-bit or 256-bit hashes. Continued reductions in the cost of computation, combined with continued progress towards quantum computers, provide many reasons to question the long-term security of 256-bit hashes. However, 512-bit hashes are clearly overkill.

NIST required a 512-bit SHA-3 option with 2^{512} preimage security to match SHA-512, not to match the security needs of the SHA-3 users. Some SHA-3 candidates also offer a faster option with 512-bit output but lower security, arguing that the lower security level is perfectly adequate. Keccak is one example, providing "only" 2^{288} security for its default 512-bit option, as discussed in the previous section. CubeHash is another example, providing "only" 2^{384} security for its default 512-bit option.

Every SHA-3 candidate uses *at least* 1024 bits of RAM to reach a 2^{512} security level. CubeHash uses exactly 1024 bits at this security level (and at other security levels, for interoperability), with a very small block size and therefore many rounds per byte. Reducing the expected security level from 2^{512} to 2^{384} allows CubeHash to increase its block size to 256 bits, making it much faster.

Pushing Keccak from 2^{288} security up to 2^{512} security decreases the block size from 1024 bits to 576 bits. Keccak sacrifices only about a factor of 2 in speed here, because Keccak maintains a large block size—but this forces Keccak to have a very large state, 1600 bits. I don't like this approach: it improves speed for users who want an insane security level, but it punishes users who need to hash on small platforms, and I think it's obvious that the second group of users is much more important.

Luffa takes a different approach. It has only one 512-bit option, Luffa-512, providing 2^{512} security. It also has a 256-bit option, Luffa-256, that fits into relatively small hardware (although not as small as CubeHash!). I also don't like this approach: it creates severe interoperability problems.

The second-round CubeHash proposal has two 512-bit options, both supported by the same very small hardware but providing different security/performance tradeoffs. This document maintains those options, except for reducing the finalization as discussed above. My model is that 99% of all users will be satisfied with the speed of either option, and that all sane users will be satisfied with the security level of either option; but the higher-

speed option will be important for 1% of users, and the higher-security option might be important to meet formal checklists. The only dissatisfied users will be users who need high speed at an insane security level.

The second-round CubeHash proposal also has two 384-bit options. Both options provide the same security level against all attacks known, but one of them makes narrow-pipe attacks even more difficult. I've decided to discard the slower option; anyone who wants that type of overkill should move up to the 512-bit hash.

Faster MAC: 1 extra option. HMAC automatically turns each CubeHash option into an HMAC-CubeHash option. This document supplements this HMAC spectrum with a faster MAC option (a prefix MAC, as discussed earlier): CubeMAC128, with a 512-bit key and 128-bit output.

There is no need in the foreseeable future for a MAC longer than 128 bits. Communication is fundamentally more expensive than computation. An attacker with a billion parallel super-high-speed network connections, flooding each network connection with a billion forged packets per second for a million years, will still have only one chance in ten million of bumping into a correct 128-bit authenticator.

PRF applications also do not need longer outputs. The PRF strength of CubeMAC128 is not limited to the preimage security of CubeHash128: finding a single 128-bit preimage is of no use for an attacker trying to break a PRF with a 512-bit key. The attacker needs to perform a massive brute-force search to find a key compatible with several CubeMAC128 outputs.

Applications that need more than 128 bits of PRF output can use CubeMAC128 repeatedly on varying inputs. I have considered specifying CubeMAC256 to accelerate these applications, but applications that need a very fast PRF will be better served by a separate stream cipher.

Avoiding bait and switch. CubeHash has always had an extremely straightforward performance profile: all options use the same hardware area, and all recommended options run at the same speed. The only deviation is that CubeHash becomes much slower when it is pushed to an insane, non-recommended security level.

JH and Shabal also have flat performance profiles across all recommended options. All other SHA-3 candidates have performance variability between their lower-security recommendations and their higher-security recommendations. Many of those candidates have already engaged in clear-cut cases of bait-and-switch advertising.

For example, the Skein team has frequently advertised the small hardware area required for Skein-256 (i.e., Skein-256-256), and the speed of Skein-512, but many of these advertisements do not mention how slow Skein-256 is or how large Skein-512 is. When an independent evaluation team at the Second SHA-3 Candidate Workshop reported the slowness of Skein-256, the Skein team objected that Skein-512 should have been used instead, and convinced the authors to eliminate Skein-256 from their report. The Skein web

site nevertheless continues to say "Small devices, such as 8-bit smart cards, can implement Skein-256 using about 100 bytes of memory."

Avoiding complexity of cryptanalysis. CubeHash has already been widely recognized as an exceptionally simple hash function, easily accessible to cryptanalysts. The 8 CubeHash options together are much easier to analyze than a single option in a typical SHA-3 submission.

CubeHash offers cryptanalysts a wide range of reduced-round variants and increased-block-size variants, far beyond the limited range of options proposed for SHA-3. These extra variants do not make CubeHash more complicated to analyze; they make CubeHash *easier* to analyze, by providing a way to measure and focus cryptanalytic progress.

The different CubeHash output lengths have no effect on message processing. The IV depends on the output length, but analyzing the merits of 4 IVs is not significantly easier than analyzing the merits of 5 or 6 or 7 IVs. The final state is simply truncated to the desired output length, so shorter-output cryptanalysis is simply a restricted type of longer-output cryptanalysis. This restriction has no effect on "internal" cryptanalysis, and "external" cryptanalysis has naturally focused on the easiest case of 512-bit output, so adding 160-bit and 128-bit options will again have no effect. Note that CubeHash does not attempt to use a smaller number of rounds for a smaller output size.

The two 512-bit CubeHash options vary in message processing: one of them uses a 32-byte block, while the other uses a 1-byte block. The 1-byte block can be viewed as a 32-byte block in which 31 bytes are required to be 0, so 1-byte cryptanalysis is simply a restricted type of 32-byte cryptanalysis. Cryptanalysis of CubeHash, starting from the original submission, has consistently treated the block size as a variable.

MAC analysis is indisputably extra complexity on top of collision analysis, preimage analysis, etc., but it is not avoidable complexity: all SHA-3 candidates are required to support HMAC. The new CubeMAC can be viewed as the first stage of HMAC-CubeHash, and analysis of that stage was already a natural stepping-stone to analysis of HMAC-CubeHash.

8 Specification of the new CubeHash proposal

This document defines

- "CubeHash128" as CubeHash16+16/32+32–128 (i.e., CubeHash with 16 initialization rounds, 16 rounds for each 32 bytes of input, 32 finalization rounds, and 128 bits of output);
- "CubeHash160" as CubeHash16+16/32+32–160;
- "CubeHash224" as CubeHash16+16/32+32–224;
- "CubeHash256" as CubeHash16+16/32+32–256;
- "CubeHash384" as CubeHash16+16/32+32–384;

- "CubeHash512" as CubeHash16+16/32+32–512;
- "CubeHash512x" as CubeHash16+16/1+32–512; and
- "CubeMAC128" as CubeHash16+16/32+32–128 where a 512-bit (not 128-bit) secret key is prepended to the input.

This document further proposes CubeHash for SHA-3, specifically with the following standard options:

- "AHS128" defined as CubeHash128;
- "AHS160" defined as CubeHash160;
- "AHS224" defined as CubeHash224;
- "AHS256" defined as CubeHash256;
- "AHS384" defined as CubeHash384;
- "AHS512" defined as CubeHash512;
- "AHS512x" defined as CubeHash512x; and
- "AMAC128" defined as CubeMAC128.

9 Expected strength

The following table shows the expected cost of preimage attacks and collision attacks against each of the CubeHash hash-function proposals. All costs are expressed in bit operations (including no-ops spent by idle transistors in parallel circuits), or in qubit operations for quantum computers. Each exponent contains a "$+\epsilon$" to account for issues such as the number of bit operations required to evaluate a CubeHash round, the number of attack repetitions required for success, etc. Minor attack optimizations often affect ϵ but make no difference in the big picture.

function	preimage	collision	post-quantum preimage	post-quantum collision
CubeHash128	$2^{128+\epsilon}$ bit ops	$2^{64+\epsilon}$ bit ops	$2^{64+\epsilon}$ qubit ops	$2^{64+\epsilon}$ qubit ops
CubeHash160	$2^{160+\epsilon}$ bit ops	$2^{80+\epsilon}$ bit ops	$2^{80+\epsilon}$ qubit ops	$2^{80+\epsilon}$ qubit ops
CubeHash224	$2^{224+\epsilon}$ bit ops	$2^{112+\epsilon}$ bit ops	$2^{112+\epsilon}$ qubit ops	$2^{112+\epsilon}$ qubit ops
CubeHash256	$2^{256+\epsilon}$ bit ops	$2^{128+\epsilon}$ bit ops	$2^{128+\epsilon}$ qubit ops	$2^{128+\epsilon}$ qubit ops
CubeHash384	$2^{384+\epsilon}$ bit ops	$2^{192+\epsilon}$ bit ops	$2^{192+\epsilon}$ qubit ops	$2^{192+\epsilon}$ qubit ops
CubeHash512	$2^{384+\epsilon}$ bit ops	$2^{256+\epsilon}$ bit ops	$2^{192+\epsilon}$ qubit ops	$2^{192+\epsilon}$ qubit ops
CubeHash512x	$2^{512+\epsilon}$ bit ops	$2^{256+\epsilon}$ bit ops	$2^{256+\epsilon}$ qubit ops	$2^{256+\epsilon}$ qubit ops

The entries in this table were calculated as follows:

- Preimage attacks: Short-output attacks (exponent h) are usually best, except that narrow-pipe attacks (exponent slightly above $512 - 4b$) are best against CubeHash512.

See the "Complexity of generic attacks" appendix in the original CubeHash submission for details.

- Collision attacks: Short-output attacks (exponent $h/2$) are better than narrow-pipe attacks for all of these proposals.
- Quantum preimage attacks: Half the preimage-attack exponent (as in the second-round CubeHash submission), following the attacker's most optimistic view of what Grover's quantum algorithm can achieve.
- Quantum collision attacks: Minimum of the above exponents.

A frequently quoted 1998 paper by Brassard, Høyer, and Tapp claims that its quantum collision attack achieves exponent $h/3$. That claim is incorrect and is not reflected in the above table. See my paper "Cost analysis of hash collisions" for further discussion of this issue.

CubeMAC security. The following table shows the expected cost of PRF attacks and MAC attacks against CubeMAC128. The cost of PRF attacks (distinguishing outputs from uniform) is measured in bit operations, or qubit operations for quantum computers, as above. The cost of MAC attacks (forging a MAC) is measured in bits transferred. Of course, PRF attacks can also be used to break the MAC; the attacker's overall success probability combines the PRF attack success probability (computation compared to the table below) with the MAC success probability (network transmission compared to the table below).

function	PRF	post-quantum PRF	MAC
CubeMAC128	$2^{256+\epsilon}$ bit ops	$2^{192+\epsilon}$ qubit ops	$2^{128+\epsilon}$ bits

The entries in this table were calculated as follows: the PRF security of CubeMAC128 is expected to at least match the collision security of CubeHash512; the 128-bit output length reduces MAC security to $2^{128+\epsilon}$.